The Osborne 1 Compendium

A complete reference, assembled from the project's own research

o1next

Contents

The machine 2
Overview and history 2
Mainboard and the schematic set 2
Memory and I/O map 14
ROM and firmware 27
ROM variants and third-party ROMs 39
ROM 1.3: the source disk, and what is left 50
Reading the monitor ROM off a real machine 61
OZROM 1E: a third-party monitor ROM 66
ROM 2.0: CP/M resident in the ROM 79
Video and the character generator 114
Interfaces 115
The floppy subsystem and its formats 115
capacity 117
The floppy adapter design 124
Serial and the modem 139
The IEEE-488 port 146
Keyboard and printer 155
Peripherals and storage 156
Drive C: the IEEE-488 RAM disk 156
The Drive C reproduction: hw + firmware 173
The Drive C's Executive variant 182
The hard-disk products 190
The hard-disk decision: the ACB-4000 194
The RT-60A real-time clock 216
the O1 cell 222
CoPower-88: the host protocol 223
CoPower-88: the schematics 226
80-column boards: SCREEN-PAC to TRA 229
Software 243
The software library 243
Boot disks 243
The project 243
Emulation and testing 243
The interposer and its virtual peripherals 255

Prior art: Z80 bus interposers 263
Open questions 272
Appendices 286
Archive provenance 286
The machines: a photographic survey 311
The Portable Companion extract index 314

DRAFT. This book is assembled from the project's working notes, which are explicitly partial: chapters carry a status and some are not written yet. Where a chapter is incomplete, the source document says so and says why. The repository is the authority.

The machine

Overview and history

(not yet written)

Mainboard and the schematic set

Status: Solved. The archive item is indexed sheet by sheet, and every pinout the project builds against is read off the scan at native resolution and then cross-checked against the same board running under MAME: the Z80 socket (§4), the character-generator ROM (§4a), the ACIA clock and baud divider (§4b), the P8 floppy header (§5), and the machine-level oracle for the memory and I/O map (§6, §7, §7a). The one thing not extracted is the board's discrete glue logic — above all the wait-state generator on sheet 2/9, which is what MAME does not model (§8, §9).

`research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf` is the primary source for the O1′s logic board. It is read one region at a time in a PDF viewer at native resolution: the scan is dense, the PDF's own text layer is unusable, and the 12 pages are actually **two different drawing sets**, which is the first thing to get right (§1). Serves #62 and #64.

1. What the file actually contains

12 pages. Each page is a single 1-bit JBIG2 scan at roughly 600 dpi (6582 × 5137 px on a 785 × 610 pt sheet) — genuinely high quality; every pin number is legible at native resolution.

Each page *also* carries a text layer, but it is an early-1990s OCR pass and is garbage: the disc-controller sheet's title reads `J'U~ I 'IIS IO/OO-tOOnl`. Ignore it. (It is left in the PDF; the fresh pass in §3 is the one to search.)

The 12 pages are **two drawing sets**:

Pages	Drawing	Rev	Sheets
1–3	**DISK ELECTRONICS**, DWG 1A3004-00/01	C	1–3 of 3
4–12	**MAIN PC BOARD**, DWG 1A2011-00	E	1–9 of 9

This distinction matters. The disk-electronics set (1A3004) is the *drive's own* board — read amplifier, data separator, write driver, head connectors. The mainboard set (1A2011) is the machine this project extends. Cite 1A3004 only for the drive: the **P8 floppy connector pinout is on 1A2011 sheet 9 of 9**, and 1A3004 does not carry it (§5).

`research/README.md` describes the split as "sheets 1–3 are the disk analog board (DWG 1A3004), sheet 4+ the main logic board" — correct in outline; the exact drawing numbers and sheet counts are as tabulated above.

2. Sheet index

Titles are the drawing's own title-block text (sheet 1 of the mainboard set has no block title; its function is read off the circuit).

PDF page	Drawing / sheet	Function	Devices read off the sheet
1	1A3004 1/3	Disk read electronics: head pre-amp and peak detection, ALC, head connectors P1/P2	U9, U10, U15, U16, U18, U19, U20
2	1A3004 2/3	Disk data separator: index/sector, read data, drive connector P2	U8, U12–U22
3	1A3004 3/3	Disk write driver, erase, motor control, ±12 V/+5 V rails	U1, U2, U3, U6, U11, U12, U13
4	1A2011 1/9	CPU clock (4 MHz), reset, data buffers, power entry	UD11, UD13, UE3, UE20, UC7, **UC11 = 74LS161** clock counter, UA6, UA8, J5, P6 (battery)
5	1A2011 2/9	Address decode, ROM/RAM/I-O select, WAIT generation	UC1, UC2, UD1, UD2, UD3, UD9, UD12, UE3, Q1/Q2 (reset timing)
6	1A2011 3/9	**CPU**: Z80-A, monitor EPROM, address latches	**UC11 = Z80-A**, **UD11 = 2732**, UD4, UD6–UD10, UE6, UE10, UE11, UA4
7	1A2011 4/9	**RAM** array, address multiplex/buffer, **character generator ROM and its serialiser**	**UA15 = 2716** chargen ROM and **UA14 = 74166** pixel serialiser (§4a), **UA18 = 74LS273** character latch (drives UA15 A0–A6), UA17/UA18 and the UA23–UA48 array (2114/2112 static RAMs per sheet 1′s parts list), UE13–UE15 (74LS244), UE16–UE18 (74LS158),
10	1A2011 5/9	[illegible]	UE25 (74LS[illegible]), UD[illegible] (74LS374), [illegible]

Sheets 4–12 are numbered 1–9 of 9 in their title blocks; the mainboard set is complete in this scan.

> **Read the device lists at native resolution before citing them.** The *functions* in the last column are reliable; the part numbers came from a whole-sheet pass and at least two are wrong (§3). `UC11` in particular is listed twice — as the Z80 (sheet 3/9, verified at 600 dpi) and as a 74LS161 (sheet 1/9, overview pass). **The Z80 is the verified one.**

3. How to read a sheet

Whole-sheet views are useless — scale a 6582 px sheet onto a screen and the pin numbers vanish. Read it one region at a time at native resolution. Two tools:

```
# browse: an overview plus a grid of native-resolution tiles for every sheet
tools/schematic_atlas.sh research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf
#   -> build/atlas/index.md          tile map: page -> tile -> pixel box
#   -> build/atlas/sheet-05/r1c0.png native-res tiles you can just open

# then re-render any region exactly, once the index says where it is
tools/schematic_render.sh research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf /tmp/s
\
    --sheet 5 --dpi 600 --crop 2600 1250 1450 1050
```

`tools/schematic_atlas.sh` writes into `build/atlas/` — gitignored, because the atlas is regenerable and the *coordinates* are the durable part. Its index maps every tile to the pixel box it covers, so a region found by eye once can be re-rendered exactly later. `--info` on the render tool prints the sheet size in pixels at a given dpi. Both need poppler (`brew install poppler`), which `tools/ocr_pdf.sh` already depends on.

Read part numbers at native resolution, never from an overview. At 300 dpi `LS00` and `LS08` are indistinguishable, and so are `LS138` and `LS139`. An overview pass over this scan got both of those wrong on sheet 2: at 600 dpi it plainly reads **`UC1 LS00`** (quad NAND) and **`UB1 LS139`** (dual 2-to-4 decoder — its pin numbers 15/14/13 for enable/A/B and 12/11/10/9 for Y0–Y3 are exactly the second half of a ‘139). The §2 device lists came from that overview pass, so treat the *functions* as reliable and the part numbers as needing a native-resolution check before anything is built on them — the `UC11` conflict in §2 is one of them, and the Z80 reading is the verified side.

`research/osborne1/OCC1_1A2011-00_Schem_RevE.ocr.txt` is a fresh OCR pass (per the `research/README.md` OCR convention) so the scan is greppable for component references. It is useful for *finding* things — “which sheet has the MB8877?” — and not trustworthy for pin-level detail: tesseract on native-resolution tiles mangles exactly the tokens that matter (`UC1 LS139` came back as `uct LSiss`, `LS32` as `L$3e2`, `LS00` as `tS0e`) and a 1200 dpi pass returned nothing usable at all. Everything in §4, §4a, §4b and §5 was read by eye at 600–1200 dpi instead.

4. Extracted: the Z80 socket (UC11) is a standard Z80 DIP-40

Read from sheet 3 of 9 (PDF page 6) at 600 dpi. The symbol is drawn as **UC11, Z80-A**.

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	A11	11	+5 V	21	RD	31	A1
2	A12	12	D2	22	WR	32	A2
3	A13	13	D7	23	BUSAK	33	A3
4	A14	14	D0	24	WAIT	34	A4
5	A15	15	D1	25	BUSRQ	35	A5
6	CLK	16	INT	26	RESET	36	A6
7	D4	17	NMI	27	M1	37	A7
8	D3	18	HALT	28	RFSH	38	A8
9	D5	19	MREQ	29	GND	39	A9
10	D6	20	IORQ	30	A0	40	A10

Read directly off the sheet: A1 = 31 … A10 = 40, A11 = 1 … A15 = 5, A0 = 30, D0 = 14, D1 = 15, D2 = 12, D3 = 8, MREQ = 19, RFSH = 28, IORQ = 20, WR = 22, M1 = 27, RD = 21, +5 V = 11, GND = 29, BUSRQ = 25, NMI = 17. The unread rows are the standard Zilog DIP-40 pinout, consistent with every pin that was read.

This closes the Z80 half of `docs/research-gaps.md` gap #3 from the primary source. The O1′s socket is wired as a plain Z80 DIP-40, so any board that mates with it can use the datasheet pinout — which is what `tools/check_schematics.py` asserts, and what the interposer generators carry (#62): `+5 V` = 11, `GND` = 29, `A0` = 30, read from the databook. The sheet says exactly that.

The **other half of gap #3 is the character-generator socket**, and it is closed in §4a: `UA15` is the **2716 character-generator ROM on sheet 4 of 9** (PDF page 7, title block `RAM`), read at 1200 dpi. It is neither a 74S244 nor on sheet 5 — a wrong entry in the §2 device list that is what left #24′s second tap point unresolved.

4a. Extracted: the character-generator ROM socket (UA15) — gap #3 closed

Found at 600 dpi on sheet 4 of 9 (PDF page 7), read at 1200 dpi. The socket is `UA15`, silkscreened **2716**, drawn with the address pins on the left and the data pins on the right. The pinout is the **standard Intel 2716**, pin for pin:

Pin	Signal	Where it comes from / goes
1–8	A7–A0	A0–A6 = character code (7 bits, 128 chars) from **UA18 (74LS273)**; A7 = SCAN0
9, 10, 11, 13–17	O0–O7	the pixel row, to the parallel inputs of **UA14 (74166)**
12	GND	**bussed with pins 18 and 20, then to ground**
18	/CE	**grounded** — the ROM is permanently selected
19	A10	SCAN3
20	/OE	**grounded** — the ROM always drives its outputs
21	(A11 on a 2732)	**strapped to +5 V**
22	A9	SCAN2
23	A8	SCAN1
24	Vcc	+5 V

Address map

SCAN0–SCAN3 arrive as labelled nets (5C1 – SCANn): they are the outputs of the scan-line counters **UD17/UD18 (74LS161) on sheet 5 of 9**, which the video PIA UC15 presets from PB0–PB3. With A7 = SCAN0 and the character code on A0–A6, the address is

```
address = (scan << 7) | char       scan = {SCAN3,SCAN2,SCAN1,SCAN0}
        = 128 characters x 16 scan rows = 2048 bytes = exactly one 2716
```

The displayed cell is 8 wide x 10 high (24 rows x 10 = the 240-line video all three MAME drivers produce), so scan rows 10–15 of every character are stored but never shown.

Output path — the pixel row leaves as one serial net

O0–O7 do **not** leave the sheet as eight parallel CHAR0–CHAR7 lines. They drive the parallel inputs of **UA14 (74166)**, an 8-bit shift register on the same sheet, which serialises the pixel row and exports it as the single net VIDEO → 5D4 to the video sheet. UA14 is clocked by DOT CLK (from sheet 1/9) and its SI and /CE are wired together — the usual trick that shifts in 1s as the inter-character blank. The video sheet's own 4D1 → VIDEO reference (sheet 4, zone D1 — where UA14 sits) corroborates the direction.

This matters to #24: the char-gen socket is a **font source, not a pixel bus**. Reading the font through the tap is enough — the pixels are re-rendered from shadow VRAM.

What the strapping means for a tap

/CE (18) and /OE (20) are **hard-grounded on the mainboard**, so the ROM is always selected and always driving O0–O7. A socket tap may buffer and *read* those pins freely, but it cannot tristate the ROM to inject its own font without intercepting pins 18 and 20 between socket and ROM. Any future "supply your own font" mode needs that interception, and the pin-21 strap held at +5 V (the socket is wired 2732-style, so a 4 KiB part would fit physically).

4b. Extracted: the ACIA's clock and the baud divider (sheet 7 of 9)

Read from sheet 7 of 9 (PDF page 10) at 600–1200 dpi, prompted by two sources that describe the same modification from opposite ends — *The Portable Companion* #13 (Hogan, from inside OCC) and the Technical Manual's own Figure 4.4.

The 6850 (UC4) pin numbers as drawn. The symbol omits Vcc/Vss (pins 12 and 1), which is this schematic's house style — every symbol drops its power pins:

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
2	RXDATA	5	RTS	8	CS2	11	RS
3	RX (clock)	6	TXDATA	9	CS1	13	R/W
4	TX (clock)	7	IRQ	10	CS0	14	E
23	DCD	24	CTS	15–22	D7…D0	12 / 1	Vcc / Vss

The clock is strapped. Pins **3 and 4 are wired together** under a bracket labelled CLK — the ACIA runs both halves from one clock — and that net is driven by **UC3′s QD (pin 11)**.

UC3 is a preset-load counter, not a plain divider. It is a 74LS161 at board position **C3**, clocked (CK, pin 2) from a 4USEC/2USEC tap off the video counter through jumper **J1**, with **RCO (pin 15) inverted by UB3 (74LS04) into /LD (pin 9)** — the classic preset-load ÷N. /CLR (pin 1) is tied to +5 V and the preset inputs carry the divisor as solder-select pads. So the stock 19.2 kHz ACIA clock is (250 or 500 kHz) ÷ 13, and the 6850′s own ÷16/÷64 gives 300/1200 — or 600/2400 with J1 out.

It agrees with the Technical Manual pin for pin. Figure 4.4, "Baud Rate Hardware Modification" (manual p. 22 = **PDF page 36**; the manual's +14 offset holds, docs/o1-chargen-graphics.md), draws the same LS161/LS163 at C3 and documents the factory's faster-rate modification: **pin 2 moved** between a *"FOR 9600/2400 BAUD"* and a *"FOR 4800/1200 BAUD"* pad, jumpers **J2** and **J3**, **J1** marked **"DON'T CARE"**, and the caption box

```
300/1200 NO MOD - J1 IN
600/2400 NO MOD - J1 OUT
1200/4800 MOD W/O PIN 2 (J1 = X)
2400/9600 MOD W/PIN 2 (J1 = X)
```

bracketed *"SPEED SELECTED W/IN 6850"*. The consequence for the serial work is in docs/o1-modem-serial.md; the register-level view is in docs/o1-memory-io-map.md §"6850 ACIA detail".

The same corner of the board is also where a third-party product attaches: Advent Products' **Osbaud** solders to the ACIA's clock and supply pins (4 and 12) and two ground pins, and substitutes the clock outright — `docs/o1-modem-serial.md`.

5. Extracted: the P8 34-pin floppy connector
Read from sheet 9 of 9 (PDF page 12, DISC CONTROLLER) at 600–1200 dpi. Every pin label on the P8 symbol was read individually:

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	GND	10	DRV SEL 1	19	GND	28	WRT PROT
2	GND	11	+12 V	20	STEP	29	GND
3	GND	12	DRV SEL 2	21	+5 V	30	READ
4	**TG43**	13	+12 V	22	WRT DATA	31	GND
5	GND	14	*(not drawn)*	23	+5 V	32	SIDE SEL
6	**EARLY**	15	+12 V	24	WRT GATE	33	GND
7	*(not drawn)*	16	CLOCK (4 MHz)	25	+5 V	34	LATE
8	INDEX	17	+12 V	26	TRACK 00		
9	GND	18	DIR	27	GND		

Pins 4 and 6 are not ground. Pin 4 is **TG43** and pin 6 is **EARLY**; both are driven by the FDC (UB7) through 7406 open-collector buffers (U9/U10) and routed to the connector. Pins 7 and 14 are not drawn on P8 at all.

Two things follow, and both matter to the adapter card (`docs/floppy-adapter-design.md` §2, which carries the same pin assignment):

- **Grounding TG43 is not untidy, it is wrong.** It is the FDC's *track > 43* output, which is the drive's **write-current-reduce** input on 96 tpi drives, so clamping it low would assert reduced write current permanently.
- **The archived third-party adapter ties it low.** The Wayne Visser schematic the project archives (`research/osborne1/floppy-adapter/FloppyAdapter.sch`) ties pins 4, 6 and 7 to GND. That is safe for a passive Gotek-only adapter — a 7406 open-collector output pulled low is not damaged, and a Gotek ignores TG43 and EARLY — but it is not what the machine does. A card that switches the OEM cable through to the physical drives must not clamp two live signals, so `hardware/floppy-adapter/gen_floppy_adapter_sch.py` carries pins 4 and 6 as `FLP_TG43` / `FLP_EARLY`, passing straight through J1 ⟶ J2 like every other signal.

The 2-versus-7 discrepancy (harmless, recorded for completeness)
DWG 1A3004 sheet 1 shows connector **P2** — the drive end of the same ribbon cable — with `GND = 1,3,5,7,9,19,27,29,31,33` and `CASE = 32`, i.e. **pin 7** grounded and pin 2 not listed. Mainboard P8 shows **pin 2** grounded and pin 7 not drawn. The digits were checked at 1200 dpi against the 27/29/31/33 glyphs in the same font to rule out a misread, so the two drawings really do disagree.

It does not matter electrically: neither pin carries a driven signal on the O1 (every signal the interface has is accounted for on other pins), so both are safe to tie to ground. The adapter ties both, which is correct whichever drawing is right.

6. Running it as a machine

A scan cannot be "run": there is no vector netlist in it, only pixels, so there is nothing to hand to a simulator. Reading a schematic and *executing* the machine it documents are two different jobs. The machine itself, however, runs very well — and it is the better oracle anyway, because it exercises the design rather than our reading of it.

MAME has an `osborne1` driver, and this machine already has everything needed:

- MAME (`brew install mame`)
- the O1 ROMs in `~/Documents/Osborne1/roms/`, including `3a10082-00rev-e.ud11` — the **Rev E** monitor ROM, i.e. the same board revision as this schematic (`ROM_DEFAULT_BIOS("ver144")` in MAME's driver)
- a headless harness in the sibling project: https://github.com/Myrcurial/o1prsnt/tree/main/utilities/headless-testing-harness

That harness is a small Lua library MAME runs via `-autoboot_script`. It injects keystrokes through MAME's natural-keyboard device and reads the **screen straight out of video RAM at `0xF000`** (128 bytes/row × 32 rows), so a test asserts on what the machine is *displaying*. Run it with:

```
cd .../headless-testing-harness
O1_ROMPATH=/tmp/o1roms O1_RUNDIR=/tmp \
  ./run-test.sh mytest.lua ~/Documents/Osborne1/floppies/<image>.imd 90
```

`O1_RUNDIR` keeps MAME's `cfg//snap/` litter out of the repo. `O1_ROMPATH` must point at a directory laid out the way MAME expects — `rompath/osborne1/` or `rompath/osborne1.zip` — not a directory of loose ROM files:

```
mkdir -p /tmp/o1roms/osborne1 && cp ~/Documents/Osborne1/roms/* /tmp/o1roms/
osborne1/
```

Verified working, 2026-10-10. A three-line script (`H.boot_cpm()`, `H.dump()`, `H.finish()`) booted CP/M headless and reached `A>` in 9 emulated seconds with this on screen, read out of VRAM:

```
PIP     .COM   8k        Disk A:  1K blocks
SETUP   .COM   6k        Size=  185K, 13 Files, Used=  140K, Space=   45K
SYSGEN  .COM   2k        A>
XDIR    .COM   3k        Loading CP/M
XREF    .COM   7k        Extended Directory version 3.5
```

With `-nothrottle` the 4 MHz Z80 runs at ~20–25× real time, and because MAME counts *emulated* seconds the timings a test measures are valid 4 MHz Z80 timings. That is what makes this useful for #29: it is a machine-level oracle for the memory map, the I/O decode and bus timing, available before any hardware exists.

7. Cross-checks against the running machine

The emulator is not just a way to run the O1; it independently confirms parts of the extraction. MAME's driver (`src/mame/osborne/osborne1.cpp`) encodes the hardware, and the running machine agrees with `docs/o1-memory-io-map.md`:

- **Banks.** Driver memory map: `0x0000–0x3fff` a ROM view whose low half is the 4 KB monitor ROM `.mirror(0x1000)` and whose `0x2000–0x2fff` window is the I/O area `.mirror(0x1000)`; `0xf000–0xffff` an attribute-RAM view. That is exactly §2 of the map doc (ROM 0000–0FFF mirrored into 1000–1FFF, I/O 2000–2FFF mirrored into 3000–3FFF, video at F000).
- **Bank switching.** Driver I/O map is `map.global_mask(0x03)` with a single `0x00–0x03` write handler, dispatching exactly as §4 of the map doc says: 0 ⟶ select ROM+I/O bank, 1 ⟶ back to main RAM, 2 ⟶ map attribute RAM at F000, 3 ⟶ unmap it. The doc's four I/O writes are the four driver cases.
- **I/O decode.** The driver's window tests — floppy `(off & 0x900) == 0x100`, IEEE-488 `(off & 0x900) == 0x900`, keyboard `(off & 0xa00) == 0x200`, serial `(off & 0xa00) == 0xa00`, video PIA `(off & 0xc00) == 0xc00`, ScreenPac `(off & 0xc00) == 0x400` — match the doc's `2100/2900/2200/2A00/2C00` bases and show the decode is *partial*: peripherals alias widely and their outputs are ANDed together when more than one responds, which is why the doc calls it a "memory-mapped I/O window" rather than a decoded bus.
- **Floppy controller variant (gap #11).** The doc says "Fujitsu MB8877 ≡ WD1793 (UB7)". MAME's driver carries the same uncertainty as a source comment: *"Schematics specify a WD1793 floppy controller, but we're using the Fujitsu equivalent MB8877 here. Is it known that the original machines used one or [the other]?"* Two independent readings of the same drawing, same conclusion — the schematic's symbol is a WD1793-compatible part, the fitted chip may be either.
- **Rev E alignment.** The BIOS MAME loads by default for this machine is `3a10082-00rev-e.ud11` (BIOS 1.44), the Rev E monitor ROM — so the emulator, the archived ROM and this schematic are the same board revision.

7a. Asking the machine directly — `tools/check_o1_map.sh`

Reading the driver is still reading someone else's model of the hardware. `tools/check_o1_map.sh` boots the machine headlessly and interrogates it over its own bus, then asserts what `docs/o1-memory-io-map.md` claims. RAM is told from ROM by writing back the complement of the byte that was there and restoring it — a RAM cell follows, a ROM cell does not. Eleven checks, all passing:

```
$ tools/check_o1_map.sh
program space: mask=0xFFFF width=8   io space: mask=0x0003 width=8
booted to A> after 9.0 emulated seconds
PASS  B1    CP/M reaches the A> prompt in the emulated machine
PASS  C1    0xF000 behaves as RAM (video RAM is in the RAM bank)
PASS  C2    0x4000-0xEFFF is RAM (176/176 pages)
PASS  C3    I/O write to port 0x00 selects the ROM bank
PASS  C4    I/O write to port 0x01 selects the RAM bank
PASS  C5    the data byte on a bank-switch write is ignored
```

```
PASS  C6    I/O ports 0x04/0x05 alias 0x00/0x01 (only A0/A1 decoded)
PASS  C10   the selected bank is re-derived on instruction fetch, not latched
PASS  C7    I/O port 0x02 maps the 1-bit bank-3 plane over 0xF000
PASS  C8    ROM at 0x0000-0x0FFF mirrors 0x1000-0x1FFF
      SCREEN-PAC: not fitted (2400h bit 2 HIGH)
PASS  C9    nominally-dead bank-2 addresses alias real devices (sloppy decode)
PASS  D1    the Z80 executes in the ROM while CP/M is running
== 12 passed, 0 failed ==
```

C9 asks the machine whether the SCREEN-PAC is fitted before it decides what an alias should read, because a machine with the board answers `2400h-27FFh` as well and pulls bit 2 low there (`docs/mame-emulation.md` §2). On this machine the line reads *not fitted*, so the two reads are compared exactly.

The page maps it prints are §2 of the map doc in machine form: in bank 2, ROM across `0000-1FFF`, devices across `2000-3FFF`, RAM from `4000` up; in bank 1, RAM across all 64K. Four things it established beyond the tabulated map:

- **The bank latch is real but does not persist.** `OUT (1),A` puts the machine in the RAM bank immediately, and 10 ms of emulated time later it is back in the ROM bank — the bank is re-derived on every instruction fetch from the M1/IRQACK flip-flops, and the video PIA interrupts at 60 Hz. The map doc §4 says the selection is "qualified by M1 and IRQ-acknowledge conditions"; this is what that means in practice, and it is why a naive peek at `0x0000` while CP/M runs returns ROM bytes. Interposer logic that assumes a static bank state is wrong.
- **The I/O space decodes two address bits** (`io space: mask=0x0003`), so all 256 Z80 I/O ports are `0x00-0x03` repeated 64 times — the map doc's "only the two low address bits are decoded", measured.
- **The ROM is live while CP/M runs.** Sampling the Z80's PC during a working `DIR` finds it inside the ROM (`0x0377-0x0387`, `0x0EF7-0x0EFC`) as well as in RAM, so the ROM is paged in and out during normal operation; "the ROM is only used at boot" is not true.
- **Bank-2 aliasing is measurable, not theoretical.** `0x2500` reads the floppy register that `0x2100` reads, `0x2601` the keyboard row that `0x2201` reads, `0x2c04` the video PIA port that `0x2c00` reads. Addresses that decode to two devices at once return the AND of both — `0x2301` is keyboard *and* floppy, `0x2f00` is video PIA *and* serial ACIA.

8. Can the schematic be turned into something runnable?

Automatically, no — and that is a property of the file, not of effort. Every page is a single 1-bit JBIG2 raster at 600 dpi:

```
$ pdfimages -list research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf
page   num  type   width height color comp bpc  enc  x-ppi y-ppi size
   5     0 image    6582  5137  gray    1   1  jbig2  600   600 69.1K
   6     0 image    6592  5150  gray    1   1  jbig2  600   600 86.2K
   ...
```

No vector data at all — no lines, no symbols, no netlist, nothing to walk. The only text object is a `HiddenHorzOCR` layer from an early-1990s pass that renders a sheet title as `J'U~ I 'IIS IO/OO-tOOnl`. Whatever "extraction" means here, it starts from pixels.

Re-OCRing the labels does not work either (§3 has the mangled tokens). A harvested label index would mislead more than it helped, so the reading stays manual and §3 is about making that fast and accurate.

The honest route to a netlist is manual capture, and the only part worth capturing is the discrete glue. Most of these sheets are Z80, DRAM, FDCs and PIAs whose behaviour is already known; the unknowns are the small-scale logic — above all the **wait-state generator on sheet 2/9** (`UC1`, `UC2`, `UD1–UD3`, `UD9`, `UD12`, `UB1`, `UB7`, `UB12`, `UE1–UE3`, `UA4`, `UA5`, Q1/Q2), which is what MAME does *not* model and what issue #29 needs. Capturing that one sheet into KiCad gets ERC and netlist checking from the tooling `hardware/` already has; capturing it into Verilog additionally lets it be *run*.

What "run it as a machine" can mean, in order of fidelity:

Approach	Status	What it validates
MAME, booting CP/M headless	works (§6)	the machine, behaviourally
`tools/check_o1_map.sh` asserting the documented map	works (§7a)	our *reading* of the map, against the machine
Hand-captured glue logic as Verilog, simulated	not started	the schematic itself — the wait states MAME omits
The above, driven by bus traces logged from MAME	not started	the same, against real Z80 traffic
Whole-board gate-level simulation	not sensible	MAME already is the machine

The last two rows are a design, not a wish: MAME's Lua can log every read and write through an address space, so a captured decode/wait model can be driven with real traffic and its chip-select and `WAIT` outputs compared against what the machine actually did. A disagreement is then either a misreading of the schematic or a MAME shortcut, and both are worth knowing. It is a multi-hour careful job, and doing it badly would produce a wrong document — the failure mode this whole exercise exists to avoid.

9. What is not extracted yet

- **What the `CHAR0–CHAR7` nets carry.** They cross to the video sheet carrying a `4C1` reference (sheet 4, zone C1 — the RAM array), so they are *not* the char-gen ROM's outputs (§4a). Presumably the character-code bus out of video RAM; not yet read.

- **What UC16–UC21 do on the video sheet.** The pixel row is serialised by UA14 on sheet 4 (§4a), so the six shift registers §2 lists on sheet 5 need a second look before #24 relies on the phrase "character shifters".
- **Sheet 4 (1A2011 1/9) parts list.** That sheet carries the board's REF/DESCRIPTION table. It is partly legible in the scan and is the fastest route to a full IC inventory.
- **Per-sheet component inventories** for sheets 4, 5 and 8, which §2 lists only from a first pass — see the part-number caution in §3. (Sheet 7′s serial area is now read at pin level — §4b.)
- **The wait-state generator (sheet 2/9)** as a captured, simulatable model (§8).
- **A net-level extraction** of the whole board — not achievable from this scan without manual capture.

Memory and I/O map

Status: Solved. The port map, the three banks and the bank-switch mechanism are confirmed twice over — read from the schematic and the ROM source, then measured on the running machine (`tools/check_o1_map.sh`, #64). What this document does not carry is the decode *granularity* below a 256-byte page: the chip selects check A11 plus one of A8/A9/A10, which is a schematic-level fact recorded in `docs/o1-mainboard-schematic.md` §7 and §7a.

The machine's 64 KB address space is **overlaid by three banks** (§2), selected by two latched signals. **Every onboard peripheral is memory-mapped inside the bank-2 window `2000–2FFF`** (§3). The bank switch is **four writes to I/O ports `0x00–0x03`, whose data byte is ignored** (§4). Three IRQs and the front-panel RESET reach the Z80 (§5). §6 is the pinout reference for every socket and connector a tap or a plug has to mate with; §7 is the free-window map and the recommended virtual-peripheral base; §8 is what the map decides.

Sources: the OCC mainboard schematic `1A2011-00` Rev E (`research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf`), the Osborne 1 Technical Manual `2F00153-01` (1982), the ROM 1.44 / CBIOS source, and the MAME `osborne1.cpp` driver as an independent cross-check.

Serves #40, #62, #64 and #72. Unblocks: #11 (hardware), #16–#24 (firmware), the memory-resident vs pure-I/O decision (gap #15), the 6850 ACIA modem (#21, gap ##10), the WD179x floppy interposer (#20, gap #11), and the ScreenPac personality (#24).

---

1. CPU and clocking

Item	Value
CPU	NEC μPD780C (≡ Zilog Z80A), DIP-40, location **UC11**
Crystal	15.9744 MHz (X1)
CPU clock	4 MHz (÷4 via UD3 LS161)
Dot clock	8 MHz
Memory / char clock	2 MHz / 1 MHz
Bus width	8-bit data (DATA0–7), 16-bit address (ADR0–15)

Wait-state generation: RAM accesses insert wait states (≈375 ns for non-M1 cycles, 188 ns for the first M1); ROM executes without added delay.

1a. One crystal, no free lunch — and the 2× "business turbo"

Per Lee Felsenstein's rule of thumb — *"the number of crystals in a machine design is equivalent to the number of designers"* — the O1 has **one** oscillator, and the whole timing economy (video, ACIA baud, floppy data separator, DRAM refresh, software delay loops) is divided down from it. Consequences:

- **Serial quirk already visible:** the J1 600/2400 jumper is ~8.3% fast (650/2600 actual) because the video-tap ÷13 ÷(16|64) chain can't land exactly on 600/2400 from 15.9744 MHz. The factory's own alternate baud rate shows the seams.
- **No clean general turbo:** 3× has no integer relationship to the video chain (breaks video), and any CPU speedup drags the ACIA baud rate, the FDC data separator, and software timing loops along with it.
- **The clean-ish 2×:** doubling the CPU clock to 8 MHz keeps the ÷2 dot-clock tap coherent (15.9744 ÷ 2 = 8 MHz), so **video timing survives**. It is viable *for business apps* if the three side effects are engineered around:
  1. **Floppy** — the WD1793 data separator is timed for 4 MHz; in turbo mode physical floppies are out. Use the interposer's storage instead (#20/#22: image-library / IEEE-488 storage answers on the bus at whatever speed the host runs), or gate turbo off when the FDC is selected.
  2. **Timing loops** — accept that games/delays run double-speed; scope the feature to calculation-bound work (SuperCalc recalc, dBASE II, compilers, assemblers). A classic PC-era **"TURBO" switch** selects 4 MHz ↔ 8 MHz.
  3. **Serial** — no clean fix at the ACIA. Mitigation: build a **baud-rate buffering corrector** into the WiFi modem (#21). The modem's RP2354B talks to the far end at a rock-solid standard rate (2400/9600) and tolerates whatever the O1 ACIA is actually doing: on **send** it buffers until a batch accumulates before transmitting; on **receive** it buffers a frame, then re-clocks it out to the O1 at the rate the O1 expects. The O1-side rate drift becomes the modem's problem, not the link's.

- **Net:** a 2× business turbo is a *documented, buildable option* (mostly on the replica #49, where SRAM removes the DRAM-refresh constraint); it is not a general speedup and doesn't pretend to be.

2. Memory banks

The 64 KB address space is overlaid by **three banks**, selected by the `ROM MODE*` signal and the `BIT 9` latch (see §4).

Bank 1 — main RAM ("normal mode")

64 KB of 4116 dynamic RAM. The top 4 KB is dual-purpose as video RAM.

Range	Contents
`0000–3FFF`	RAM (boot ROM overlays the low 16 KB when bank 2 is active)
`4000–EFFF`	RAM — CP/M TPA, BDOS/CCP, BIOS live here
`F000–FFFF`	Video RAM low 8 bits (128 cols × 32 rows char data)

Page zero. CP/M owns `0000–00FF` while a program runs, but the *Osborne 1 User's Guide* "Memory Allocation" chart (p. R-243) marks **`0050H–005BH` — 12 bytes — as "not used"**, and at least one published program relied on that: the date facility in *The Portable Companion* #7 keeps the system date there in both binary and ASCII form (`docs/rt60a-analysis.md`). Treat it as documented free scratch space — worth remembering for the interposer (#12).

Bank 2 — BIOS ROM + memory-mapped I/O ("shadow mode")

Selected when `ROM MODE*` is active (low). Only `0000–3FFF` is affected; `4000–FFFF` still reads bank 1 RAM.

Range	Contents
`0000–0FFF`	Monitor ROM (4 KB, 2732 at **UD11**; earlier 2× 2716)
`1000–1FFF`	Unused — reads as mirror of the ROM
`2000–2FFF`	Memory-mapped I/O window (see §3)
`3000–3FFF`	Unused — reads as mirror of `2000–2FFF`
`4000–FFFF`	Mirrors bank 1 RAM

Bank 3 — video attribute bit ("dim bit")

Selected when `BIT 9` latch is set. A 9th bit per video cell, stored in a dedicated 4116. Only `F000–FFFF` is affected.

Range	Contents
`F000–FFFF`	Video attribute RAM, 4K×1 (dim attribute = bit 8 of each cell)

2a. Video memory geometry — 128 columns wide, 52 displayed (#97)
The 4 KB at `F000–FFFF` is **32 rows of 128 characters**, of which **24 rows of 52 characters are displayed at a time**:

> Memory from 0F000 hex to 0FFFF hex is used for the video memory map. That memory is considered to be 32 rows of 128 characters, even though only 24 rows of 52 characters are actually displayed on the monitor screen at a time. The character matrix is 7 × 9 in an 8 × 10 box. — *Osborne 1 Technical Manual*, §5.2 "Video Memory", p. 23

This is the answer to the "64 vs 52" question in #97. The screen is a *window* onto a buffer 2.5× wider and 1.3× taller than what is shown, and the displayed width is a property of the window, not of the memory. That is why the same 4 KB shows 52 columns on stock hardware, 80 on Nuevo and 104 with the ScreenPac (`mame-emulation.md` §2) — the text layout is the CBIOS's, the window is the machine's.

It is also why the scrolling feels different from a terminal's. Holding Ctrl and using the arrow keys slides the window over the memory rather than moving the memory, which is a register change and not a rewrite of the buffer — the same mechanism `mame-emulation.md` §6 calls a ring buffer. A terminal scroll *does* move the lines, which is why that stays chunky while the viewport slide is smooth. (The Ctrl+arrow slide is a hardware observation, reported on the real machine; the geometry above is the manual's.)

Bit fields (§5.2, figure): bits 0–6 are the **ASCII character**, **bit 7 is the underline attribute**, **bit 8 is dim**. Bits 0–7 are the byte in the video RAM proper; bit 8 is the separate 4116 in bank 3. This is why the MAME probe masks with `% 128` to read characters (`tools/o1_mame_probe.lua`).

The window width is software-selectable. The SCREEN-PAC user guide (p. 13) gives the mechanism, and it is in the stock BIOS rather than on the board:

> At BIOS location **E1A5H**, POKE or store **0 for 52 columns / 1 for 104 columns / 3 for 80 columns**. CALL BIOS location **E4E5H**. … In these same formats, SCREEN-PAC provides **horizontal scrolling by 32 column blocks**.

So 52, 80 and 104 are three widths of one window onto the same 128×32 buffer, and the 32-column blocks are why the slide lands on round numbers. Without the SCREEN-PAC only 52 is available, because the wider windows need the board's dual-port video RAM — the ROM contains the routine either way.

Logical width is not physical width — this is what "64 vs 52" turns on. The SCREEN-PAC guide (p. 13) separates the two, and it is the piece the technical manual's geometry leaves implicit:

> **Logical screen size** refers to the number of characters you can enter on a line without automatically inserting a carriage return. Logical screen size can range from 1 up to a **maximum of 128 characters**.
>
> **Physical screen size** refers to the number of columns actually displayed on your screen. With SCREEN-PAC, this will be 52, 80, or 104. — *Osborne 1 SCREEN-PAC User Guide*, p. 13

So a program's line width is a **software** choice up to 128 — the width of the video memory row — while the window is 52, 80 or 104. A drawing 64 columns wide is a *logical* line of 64 characters: it fits the row, and the window slides over it. The guide also warns against the obvious mistake, which is worth knowing before writing anything that sets a width: "logical screen size should NOT be less than physical screen size", or the display scrambles.

The width is set from the configuration screen (Item C, "Screen Size", in the Osborne 1 Configuration Program) or from software by poking `E1A5H` and calling `E4E5H` as above. Both do the same thing.

The cursor is hardware, and it can be reversed (#24). The stock cursor is an **underline**, and OCC knew it was poor — Thom Hogan argued against it while at the company and was refused twice (*The Portable Companion* #13, June/July 1984, "Reverse Video and a Faster Serial Port", p. 24): he could not get the change made permanent, and could not get permission to publish it, *"as the company was worried about the legal implications of publishing modifications"*. The modification makes the cursor a **solid white block**, and the article locates the circuit precisely: a **74LS86 piggybacked on the 74LS00 at E22**, with **pin 1 of the 74LS74 at UE21 lifted**, plus changes around the VIDEO and DIM signals.

So **the cursor is generated in hardware** — not by the character generator, and not by the attribute bit — which matters to a video personality: it has to reproduce a cursor the machine makes outside video RAM, and the block cursor is the variant OCC's own engineer preferred. Extract: `research/technical-articles/TPC-Issue13-1984-pp26-31-reverse-video-and-9600-baud.pdf`.

And it works with the SCREEN-PAC fitted, leaving the board in place. Hogan's version is for a stock machine. A reader's fix, printed in *The Portable Companion* #15 (Jan/Feb 1985, letters, PDF p. 32), makes the block cursor work on a machine with the **SCREEN-PAC 80** installed — *"and it can be carried out without removing the piggy-back board containing the Screen-Pac 80 electronics"*. What changes is **where** the logic is: the modification is done on the *upgrade board's* own chips — **U2 (74HC00)** and **U23 (74HC74)** — not the main board's, and the letter warns in capitals that *"THESE PINS ARE NOT THE SAME ONES LISTED IN THOM HOGAN'S ARTICLE!"*.

That the SCREEN-PAC carries a 74HC00 and a 74HC74 of its own is useful in itself: the cursor logic is **duplicated on the piggy-back board**, which fits the service manual's description of the upgrade as an IC-harness board soldered over the main logic (`2F00040` §6.12, the source used by `firmware/screenpac-video/`). Extract: `research/technical-articles/TPC-Issue15-1985-pp32-block-cursor-with-screenpac.pdf`.

2b. The monitor RAM and the system clock (#7, #81)

The top of bank 1 RAM (`EF00–EFFF`, 256 bytes) is the **monitor RAM**: the ROM's own storage, above the CP/M TPA. The CBIOS RAM map (`occram25.asm`) and the ROM's absolute equates (`rom144.asm`) agree on its layout, and it contains the machine's **system clock**:

Address	Cell	Meaning
`EF00`	`TEM`	boot-routine scratch
`EF08`	`ROMRAM`	ROM/RAM flag
`EF60`	`ESCH`	ESC-holding flag
`EF61` / `EF62`	`PIAAD` / `PIABD`	last PIA-A / PIA-B data
`EF63` / `EF64` / `EF65`	`IDAY` / `IMONTH` / `IYR`	calendar date, binary
`EF66`	**`HOURS`**	hours, 2-digit BCD
`EF67`	**`MINS`**	minutes, 2-digit BCD
`EF68`	**`SECS`**	seconds, 2-digit BCD
`EF69`	**`SEC6`**	sixtieths, 2-digit BCD
`EF6A`	`DACTVE`	disk-active countdown
`EF6B`	`BELCNT`	bell countdown
`EF6C`	`LLIMIT`	max columns in a logical line
`EF6E`	`LDTRK`	last track used on the non-selected drive
`EF6F–EF70`	`IESTK`	saved stack pointer
`EFD0`	`SAVTYP`	disk type (density + sector size)
`EFD1`	`RWCMD`	sectors to read/write
`EFEF`	`VRTOFF`	last vertical offset
`EFF0–EFFF`	**`INTBL`**	8-word IM 2 interrupt vector table

`HOURS`–`SEC6` is a real clock. The stock ROM's 60 Hz interrupt handler (`GKEY`) calls `UPTIM`, which increments `SEC6`, `SECS`, `MINS` and `HOURS` in BCD with a 24-hour wrap. It starts at midnight on every boot, nothing sets it and nothing displays it in a stock machine — but OZROM 1E documents the same four cells and adds a setter, a display and an alarm. The `EFF0` vector table is where the two clock products hook: `INTBL+(4*2)` = `EFF8` is the keyboard-interrupt vector the RT-60A's `CLOCK.COM` replaces, while OZROM calls its own hook at `EFF2`. See `docs/rt60a-analysis.md` ("The third shape") and `docs/ozrom-1e.md`.

3. I/O decode (bank 2, "memory-mapped I/O")

All onboard peripherals are **memory-mapped**, reached through the bank-2 window. Chip-selects are generated from address lines within `2000–2FFF`.

Address	Device	Chip	Registers
`2100–2103`	Floppy disk controller	Fujitsu MB8877 ≡ WD1793 (**UB7**)	2100 stat/cmd, 2101 track, 2102 sector, 2103 data
`2201–2280`	Keyboard (read)	81LS95 + 74LS05 (**UE12/13/14**)	One address bit per row: 2201=row0 … 2280=row7; columns on DATA0–7
`2900–2903`	IEEE-488 PIA	MC6821 (**UC7**)	2900 PA data/dir, 2901 PA ctrl, 2902 PB data/dir, 2903 PB ctrl
`2A00–2A01`	Serial / modem ACIA	MC6850 (**UC4**)	2A00 status(R)/control(W), 2A01 RX(R)/TX(W)
`2C00–2C03`	Video PIA	MC6821 (**UC15**)	scroll/offset latches, bell, brightness, dim control
`2400` (read)	SCREEN-PAC presence	—	**bit 2 LOW = SCREEN-PAC fitted**, HIGH = stock machine (user guide p. 13). Read-only status bit, not a register

6850 ACIA detail (#21 modem, gap #10)

- Base `2A00`: read = status, write = control. `2A01`: read = RX, write = TX.
- Baud clock from divider **UC3** (÷13) fed by a 2/4 µs tap off the video counter (UA13). ACIA (UC4) divides by 16 or 64 ⟶ **1200 or 300 baud**.
- Jumper **J1** selects 300/1200; Rev E+ can be re-jumpered to 600/2400.
- **Faster rates, from inside OCC** (#21): *The Portable Companion* #13 (June/July 1984), "Reverse Video and a Faster Serial Port" by **Thom Hogan** — the same Hogan whose `AUTOCPM.ASM` is the source of the site wordmark (#97) — written as former OCC manager of software and publications: *"developed internally by the company, [but] not promoted … the primary … was the problem of warranty repair on modified machines."*
  - Removing **J1** doubles 300/1200 to **600/2400** — exactly as above — and he adds a detail the schematic cannot show: **the SETUP program cannot sense J1**, so it still displays 300/1200 after the modification.

  - For **2400/9600** the mod is to the baud divider: the **LS161/LS163 at C3**, with **pin 2 moved to the middle pin of J2** ("for 2400/9600 baud rates rather than 1200/4800") and pins 3, 7 and 10 lifted. Figure 3 in the article has the detail. Extract: `research/technical-articles/TPC-Issue13-1984-pp26-31-reverse-video-and-9600-baud.pdf`.
- **Pin level, from the schematic** (sheet 7/9 of 9, read at 600–1200 dpi): the ACIA's **RxCLK (pin 3) and TxCLK (pin 4) are strapped together** on the board and driven by **UC3's `QD` (pin 11)**. UC3 is a **74LS161 preset-load counter** — `RCO` (pin 15) is inverted by **UB3 (74LS04)** into `/LD` (pin 9) — so the count length comes from the preset pads rather than being a power of two; that is where the ÷13 of the stock 19.2 kHz clock lives. Pin numbers as drawn: `RXDATA` 2, `RxCLK` 3, `TxCLK` 4, `/RTS` 5, `TXDATA` 6, `/IRQ` 7, `CS2` 8, `/CS1` 9, `CS0` 10, `RS` 11, `R/W` 13, `E` 14, `D7…D0` 15–22, `/DCD` 23, `/CTS` 24 (Vcc/Vss, pins 12/1, are not drawn on the symbol).
- **The factory's own 9600/2400 modification is documented**, which the archive had only from inside OCC: *Osborne 1 Technical Manual* **Figure 4.4 "Baud Rate Hardware Modification", p. 22** (PDF page 36 — the manual's printed-page offset of **+14** holds here too, see `docs/o1-chargen-graphics.md`). It draws the LS161/LS163 at **C3** with **pin 2** moved between a *"FOR 9600/2400 BAUD"* pad and a *"FOR 4800/1200 BAUD"* pad, jumpers **J2** and **J3**, **J1** marked **"DON'T CARE"**, and the caption box `300/1200 NO MOD - J1 IN` / `600/2400 NO MOD - J1 OUT` / `1200/4800 MOD W/O PIN 2 (J1 = X)` / `2400/9600 MOD W/PIN 2 (J1 = X)`, bracketed ***"SPEED SELECTED W/IN 6850"***. That is the same modification Hogan describes from inside OCC in #13 — the 6850 choosing within each pair (÷64/÷16). It is also the "method shown on page 22 in the Technical Manual" that *The Portable Companion* #8 points a would-be 80-column terminal user at (see `docs/o1-modem-serial.md`).
- **A fifth route, and a third-party one:** Advent Products' **Osbaud** (1983, $59.95, 16 DIP-switch rates) is soldered to the ACIA's own clock and supply pins and replaces the clock net outright — see `docs/o1-modem-serial.md`.
- **The ACIA's control-register init byte is in the BIOS at `1FE9H`.** The vendor's own "Programming the serial port" patch (*The Portable Companion* #7, June 1983, PDF pp. 56–57) rewrites `1FE9H`/`1FEAH` to move off the 8-N-1 default, with a table of 6850 control codes (`55H` = 8-N-1 @1200). A second handle on the ACIA besides the port pair — see `docs/o1-modem-serial.md` §"Framing".
- Modem status (DTE): MSB (pin 4), MCB (pin 8), RI (pin 9); BIOS `LISTST` at `0E12D` returns `0FFH`/`00H`.
- **IRQ0** = ACIA interrupt (one of three IRQ sources; see §5).

WD179x floppy detail (#20 floppy interposer, gap #11)

- Controller **UB7**, registers `2100`–`2103` as above.
- Drive-select / density lines are driven from the **video PIA (UC15)** outputs (DRIVE1/DRIVE2/DDEN), **not** a dedicated latch — see schematic sheet 5. `DDEN` selects double density when the add-on DD board is present.
- Stock format: single-sided single-density, 40 tracks × 10×256-byte sectors (102 KB/drive), FM. DD option board adds 5×1024 Osborne-DD, 8×512 IBM, 18×128 Xerox-820, 9×512 DEC-182 formats.

Keyboard detail

- 8×8 keyswitch matrix; rows driven on address lines A0–A7 (one-hot), columns returned on DATA0–7.
- Read strobes at `2201, 2202, 2204, 2208, 2210, 2220, 2240, 2280` (i.e. `2200 + (1<<row)`).
- 14 programmable keys (Ctrl-0…9, four arrows) via BIOS + SETUP.

4. Bank switching (the mechanism)

Banking is controlled by **writes to I/O space**, decoded by **UD10** (LS138) on the two low address bits; the data bus value is **ignored**.

I/O write	Effect
`0x00`	Clear UB4 ⟶ `ROM MODE*` low ⟶ **select bank 2** (ROM+I/O). Also forced by CPU reset.
`0x01`	Preset UB4 ⟶ `ROM MODE*` high ⟶ **select bank 1** (main RAM), if `MREQ` low.
`0x02`	Set `BIT 9` latch (UE10A) high ⟶ map **bank 3** (dim bit) into `F000–FFFF`.
`0x03`	Clear `BIT 9` latch ⟶ map bank 1/2 back into `F000–FFFF`.

`ROM MODE*` is *also* set by hardware: power-on reset, and the NMI flip-flop chain (UB6/UB4). Bank selection between 1 and 2 is additionally qualified by M1 and IRQ-acknowledge conditions via three flip-flops — so interrupt acknowledge and the front-panel RESET reselect the ROM bank automatically.

> **Interposer note:** because only the two low address bits are decoded and data is ignored, *any* `OUT (0),A` / `OUT (1),A` toggles banks. The interposer must **never** drive I/O addresses `0x00–0x03`.

Measured on the running machine (`tools/check_o1_map.sh`, #64). All four writes behave exactly as tabulated — the *port number* selects, the data byte is ignored, and ports `0x04–0x07` alias `0x00–0x03` (MAME's I/O space reports `mask=0x0003`, i.e. only A0/A1 reach the decode). The M1/IRQACK qualification above is not a footnote: write `0x01` and the RAM bank is selected *immediately*, but 10 ms of emulated time later the machine is back in the ROM bank, because the bank is re-derived on every instruction fetch and the video PIA interrupts at 60 Hz.

Consequences for an interposer:

- The bank state **cannot be latched and assumed** — it has to be sampled per bus cycle, and a bank-2 I/O access is only meaningful while the CPU is actually running with bank 2 selected.
- At the idle `A>` prompt the machine is parked in the ROM polling loop with **bank 2 live**, so the live bank depends on what is running. That does not make the ROM readable *from CP/M*: a

program in the TPA runs in bank 1 with the ROM banked out, which is why `OUT (0)` is needed to select it and why `DUMP`/`DDT` return nothing useful (`docs/o1-rom-read-from-machine.md` §2).

5. Interrupts

Three IRQ sources into the Z80: - **IRQ0** — serial ACIA (UC4) - **IRQ1** — video PIA IRQA (UC15) - **IRQ2** — IEEE-488 PIA IRQA (UC7)

NMI is generated by the front-panel **RESET switch S1** (debounced by UE20, sequenced through UB6/UB4 on `M1`): NMI restarts the ROM program at `0000` and forces bank 2. There is **no periodic/60 Hz NMI** — the "60 SEC" signal is a video-tick line into the video PIA CA1 (used by the console routine's real-time clock), not an NMI.

6. Connector / socket pinouts (mechanical tap points)

Z80 CPU — DIP-40 (UC11)

Standard Z80A pinout (Zilog Z80 Microprocessor Family Databook; the O1 fits an NEC μPD780C, pin-compatible), and the O1′s own sheet says the same thing pin for pin — the socket is wired as a plain Z80 DIP-40, so a board that mates with it can use the datasheet table (`docs/o1-mainboard-schematic.md` §4, read at 600 dpi). `tools/check_schematics.py` asserts this table against every connector that mates with the O1′s Z80 socket.

Pin(s)	Signal
1–5	A11, A12, A13, A14, A15
6	CLK (4 MHz)
7–10	D4, D3, D5, D6
11	+5 V
12–15	D2, D7, D0, D1
16	INT
17	NMI
18	HALT
19	MREQ
20	IORQ
21	RD
22	WR
23	BUSAK
24	WAIT
25	BUSRQ
26	RESET
27	M1
28	RFSH
29	GND
30–34	A0, A1, A2, A3, A4
35–40	A5, A6, A7, A8, A9, A10

Character generator ROM — 24-pin (UA15)

2 KB (2716) character ROM at **UA15** — on **sheet 4 of 9** (PDF page 7, the RAM sheet), *not* the video sheet; see `o1-mainboard-schematic.md` §4a for the extraction. 128 chars × 8×10 in an 8×10 box (7×9 visible), addressed by (char code + scan line), output to the video shift register (UA14, 74166). This is the second tap point the ScreenPac personality (#24) uses. (It is the *third* tap on the interposer, which also leads to the main ROM socket UD11 — see `docs/virtual-peripherals-block-diagram.md` §1 and `docs/o1-rom-2.0.md` §4.1a.)

Pin	Signal
1–8	A7–A0 (`A0`–`A6` = char code from UA18; `A7` = `SCAN0`)
9, 10, 11, 13–17	O0–O7
12	GND — **bussed with 18 and 20 to ground**
18	`/CE` — **grounded**
19	A10 (`SCAN3`)
20	`/OE` — **grounded**
21	strapped **+5 V** (A11 on a 2732)
22	A9 (`SCAN2`)
23	A8 (`SCAN1`)
24	VCC (+5)

The socket is the plain Intel 2716 pinout, read at 1200 dpi (`docs/o1-mainboard-schematic.md` §4a carries the extraction). Because `/CE` (18) and `/OE` (20) are **hard-grounded**, the ROM **always drives** `O0`–`O7`: a tap may buffer and read those pins, but it cannot tristate the ROM to inject its own font without intercepting pins 18 and 20 between socket and ROM.

Other onboard connectors (for reference)

Every connector on an Osborne 1 is on the FRONT of the machine — there is nothing to plug into at the back. That is not a detail: it is why fitting a period peripheral meant unclipping the keyboard, which docks to seal that same face (the O1A design note in `hardware/next-integrated/`), and it is why every standalone device in this project attaches at the front (#47, #56, #84).

- **P3** — IEEE-488, 26-pin edge (pins 25/26 unused)
- **P2** — RS-232, DB-25 (DTE)
- **P1** — Modem
- **P4** — Keyboard, 20-pin IDC (1=GND, 2–9 rows 4,0,3,6,2,5,1,7, 10–17 cols 0–7, 18 NC, 19=+12V, 20 GND). On the later **O1A ("blue/grey")** machines the coiled-cable plug has **24 holes but only the inner 20 mate with pins**. Pin 19 carries **+12V only if jumper J6 is bridged** (a 2-pin link, open from the factory) through **R21 (22Ω)** — the intended way to power an accessory off the keyboard cable. Rows are driven low (open-collector inverters, one-hot); columns are pulled up to +5V via a 3.3K resistor pack and buffered onto the data bus. Reference: O1 Technical Manual; pinout per pinouts.ru (verified against a PCB), archived in-repo at `research/osborne1/keyboard/osborne_keyboard_pinout.html` (provenance alongside) — see #47 for the full matrix map.
- **P8** — Floppy drive, 34-pin
- **P5** — Video, 10-pin inline

7. Free I/O windows & recommended virtual-peripheral base

The bank-2 I/O window is sparsely decoded; many sub-ranges alias the real devices. Conservatively **free** (no onboard device) within `2000–2FFF`:

Range	Status
`2000–20FF`	Free — and **clean**: no device mask matches it (see below)
`2104–21FF`	Free (above floppy)
`2300–28FF`	Free (above keyboard, below IEEE) — **not all clean**: `2300–23FF` and `2500–27FF` alias the floppy and the keyboard, and `2400–27FF` is the SCREEN-PAC's own select (its presence bit is `2400h`)
`2904–29FF`	Free (above IEEE PIA)
`2A02–2BFF`	Free (above serial, below video PIA)

The O1 decode is "sloppy" (MAME): the chip selects do not check all address bits, so large regions alias real devices. Treat only page-aligned blocks *between* the listed devices as safe, and validate on hardware before committing.

Measured — the aliasing is real, and it has a 256-byte answer. The five selects check A11 plus one of A8/A9/A10 each, so of the region above only three page-aligned blocks are selected by *nothing*: **`2000–20FF`, `2400–24FF` and `2800–28FF`** (offsets `000`, `400`, `800` — A11 set with A8/A9/A10 clear, or all four clear). `0x2301` is the floppy's track register *and* the keyboard's row 0 (`docs/o1-mainboard-schematic.md` §7a), which is what moved the ROM↔Pico door from `2300h` to `2800h` (`docs/rom-pico-protocol.md` §2). The SCREEN-PAC's window test in the driver is `(off & 0xc00) == 0x400` — `2400–27FF` — so `2400–24FF` is clean of the *stock* machine but not of a SCREEN-PAC one, `2400h` being its presence bit. The two blocks free on every machine are `2000–20FF` and `2800–28FF`.

And the one device that is free by *mutual exclusivity* claims no bank-2 memory at all. Drive C — the standalone reproduction (#84) and the interposer's own `drive-c-gpib` personality — cannot be fitted at the same time as the interposer (`docs/mailbox-protocol.md` §1), which invites the question of whether its space is reusable. Checked against the characterisation (`docs/drive-c-protocol.md` §1–§2, `docs/drive-c-design.md` §2): **it is not a GPIB device and has no window.** The Drive C connector (P3) is used as a **parallel port**, and the host drives it by bit-banging the **mainboard's own** MC6821 PIA at `2900–2903` — machine hardware, always fitted, not borrowable. Nothing in the Drive C material names a `2xxx` address other than that PIA. What its exclusivity does free is the **drive-2 identity in the mailbox** (which the interposer already owns) and the **P3 connector itself** — a bus, not

an address. So there was no Drive C window to put the ROM↔Pico door in; `2800h` is inside this document's own recommended region instead, chosen from the blocks above.

Recommendation

- **Primary virtual-peripheral block: `2800–28FF`** (256 bytes). It is inside the clean gap between keyboard (`2201–2280`) and IEEE-488 (`2900–2903`) *and* it is one of the three blocks no select touches (above). The ROM↔Pico door takes `2800h`/`2801h` (`docs/rom-pico-protocol.md` §2); suggested sub-allocation for the rest: CoPower-88 emulation, network/modem FIFO, RTC, status regs.
- **`2000–20FF`** is equally clean and equally page-aligned if a second disjoint window is needed; `2400–24FF` is clean only of a stock machine, because `2400h` is the SCREEN-PAC presence bit.
- **Secondary: `2B00–2BFF`** (between serial `2A01` and video PIA `2C00`) if a third window is needed — but note that `2B00–2BFF` *does* alias the serial ACIA's select, so it is free of the device, not of the decode.
- Avoid `0x00–0x03` (bank switch) entirely.
- The interposer should decode **bank-2 I/O space** (memory-mapped) rather than Z80 `IN/OUT` port space, since onboard devices are memory-mapped and the plain I/O port space `0x00–0xFF` is almost entirely consumed by the bank-switch decoder aliases (only low 2 bits decoded → `0x00–0x03` repeats every 4 bytes across the whole low port page).

8. Decisions this unblocks

- **Gap #15 (memory-resident vs pure I/O): → pure I/O**, memory-mapped in bank 2 in the §7 window (`2800–28FF`). No need to steal RAM or fight the bank flip-flops.
- **Gap #10 (modem ACIA ports): →** `2A00/2A01` confirmed; the WiFi modem can either emulate the 6850 there or live at a FIFO in the same §7 window with a small BIOS hook.
- **Gap #11 (floppy variant): →** MB8877 ≡ WD1793 confirmed; registers at `2100–2103`; drive/density selects are PIA-driven (emulate via PIA shadow).
- **#24 (ScreenPac): →** two tap points confirmed: Z80 DIP-40 (UC11) and char-gen 24-pin (UA15). The interposer adds a **third** — the main ROM socket UD11 — for ROM 2.00 (#83); ScreenPac itself never needed it.

Sources

- OCC schematic `1A2011-00` Rev E (`research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf`), sheets 1–5 (CPU, RAM, video, IEEE-488, serial/disk).
- Osborne 1 Technical Manual `2F00153-01` (1982), ch. 3 (memory), ch. 4 (interfaces), ch. 5 (video), ch. 7 (disk), ch. 8 (keyboard), ch. 11 (theory of operations: ROM MODE/bank select, NMI).
- MAME `src/mame/osborne/osborne1.cpp` (bank/I/O map cross-check).

ROM and firmware

Status: Solved. The source is indexed and every port and vector table it holds **agrees with** `docs/o1-memory-io-map.md` §3, which had been derived from the schematic and from the running machine; and `rom144.asm` **reassembles with the vendor's own assembler to the shipped ROM, byte for byte** (*The 1.44 reassembly*, below). What is not done: the 1.41 CBIOS source has

never been diffed against the CBIOS on a shipped system disk, and the video register semantics are still read from the schematic rather than from the source (*Open questions / next steps*). **Issues:** #72, #40, #203.

Osborne's own source for the machine's firmware — the thing the rest of the project had been inferring — and the primary source for the interfaces the rest of `docs/` builds against:

Confirmed	Section
The five memory-mapped I/O bases — floppy `2100`, keyboard `2200`, IEEE-488 PIA `2900`, 6850 ACIA `2A00`, video `2C00` — agreeing with `docs/01-memory-io-map.md` §3 entry for entry	*The memory-mapped I/O map, as the source states it*
The monitor RAM map (`0EA80–0EFFF`) and the interrupt vector table at `0EFF0H`	*Monitor RAM and vectors*
The WD179x floppy command set the BIOS issues (#20)	*Floppy command set*
The IEEE-488 register programming, the GPIB command bytes, the inverted bus, and the *"IEEE always appears to be ready"* stub (#6/#22, #19)	*IEEE-488 — and the parallel printer is the same port*
The video write pairs (`+1`⟶`+0`, `+3`⟶`+2`) and `ENADIM`/`DISDIM` through `OUT 2`/`OUT 3`	*Video, as the ROM drives it*
A negative result: the firmware never selects its own bank — there is no `OUT 0` or `OUT 1` anywhere in the ROM	*Bank switching: what the ROM actually does*

It is a safety copy of material already public in several places; see *Provenance* at the end.

The two components

The machine's firmware is **two separate programs**, and the source tree has one of each:

Component	Lives in	Source here	Rev
Monitor ROM — boot, video, keyboard, disk, IEEE-488, interrupt handlers	the 2732 at UD11	`rom144.asm`	1.44
CBIOS — CP/M 2.2 character and disk BIOS	loaded into RAM from the system disk	`occbio*.asm`	1.41

They version independently, so **do not assume the 1.41 CBIOS source is what ships inside the 1.44 ROM.** The monitor ROM is the 4 KB in the socket; the CBIOS is a RAM-resident overlay assembled to a specific system size (below).

File index

`bios141.gen` is the build recipe, and it lists exactly the files present here, so the set is complete:

File	Lines	What it is
`rom144.asm`	4655	The monitor ROM, Rev 1.44, `ORG 0`. Z80. Roger W. Chapman, 2/4/1983. Title: "DOUBLE DENSITY ROM". Contains the memory-mapped I/O equates, RAM storage locations, IEEE/disk equates, monitor main loop, video, keyboard, disk drivers, the IEEE-488 interface and drivers, the parallel (Centronics) printer path, and the interrupt handlers.
`occtxt6.ast`	374	The shared equates file (`S=OCCTXT6.AST` in the build): port bases, ROM vectors, monitor RAM bases, character codes, disk-type constants. `occbio05.asm` says "*FOR USE WITH OCCTXT6.AST ONLY*".
`occbio05.asm` … `occbio95.asm`	47, 145, 80, 70, 341, 141, 246, 60, 505, 66	The CBIOS listing, split into ten sections. `05` is the title block and layout; `45` holds the disk control blocks (`DPBASE`, `DPHGEN`); `85` the I/O-byte dispatch and character devices.
`occram15.asm`, `occram25.asm`	31, 175	"Monitor RAM Storage" — the RAM-resident data area, `ORG MRAM`, assembled into both the ROM and the CBIOS.
`bios141.gen`	56	The generation procedure: needs ACT 3.5e, DDT 2.2, SYSGEN 2.1 and MOVCPM, and the twelve files above.
`tran.txt`	61	The 1.44 ROM release memo (Sai Kit, 2/4/1983): how the HEX and print files were generated, on a DD machine with a Corvus hard disk.
`release.txt`	23	The 1985 enhancement letter (Barth/Hall/O'Connor, 1 Oct 1985) granting permission to reproduce the code and BIOS listing. Retained verbatim — see `research/README.md`.
`bios144.asm`	5230	The 1.44 CBIOS, `ORG 0`. Z80. Roger W. Chapman, 2/4/1983. Title: "DOUBLE DENSITY CBIOS". Contains the boot loader, the CP/M BIOS jump table, the character-device routines, the disk-I/O routines, and the disk parameter blocks.
`bios144.hex`	3030	Intel HEX of the 1.44 CBIOS, for loading without assembling.

Revision lineage

Monitor ROM 1.44 — the header carries the delta from 1.43, which is the useful part (it is a change log for the DD machine):

```
CBOOT  - added seek 10 tracks and home drive : tsk
GKEY   - fixed bell bug                      : tsk
SENDEN - changed the number of retrys to NRETRY
PSEKC  - added disk head settle delay
SELDRV - changed start up delay to 500 ms    : tsk
```

`NRETRY = 10` (disk retry count), `SLD_RCT = 3` (slide-key repeat), and `BRTBIT`/`DIMBIT` (`80h`/`00h`) are defined at the top of the file. The 1.44 image is `research/roms/os1-144.rom` (CRC32 `c0596b14`) — see `docs/o1-rom-variants.md`.

CBIOS 1.41 — from `occbio05.asm`/`occbio15.asm`:

```
MSIZE = 59              ; 59K system
CCP   = 0CB00H          ; location of the CCP
BIOS  = ccp+1600h
BDOS  = ccp+806h
DDT R (relocation value) = 1F80H-BIOS
```

So the CBIOS is relocated for a 59K system, and `occbio15.asm` carries the CP/M disk-type tables — `HSTSIZ = 1024` (blocking/deblocking buffer), `FPYSIB = 2048/128` (SD block = 2K), `FPYDIB = 1024/128` (DD block = 1K), `DSKS1 = 5` (single density, 256-byte sectors, single sided).

The memory-mapped I/O map, as the source states it

Straight out of `rom144.asm` (`;MEMORY MAPPED I/O`, around line 83) and `occtxt6.ast`:

Address	Name in source	What it is
2100H	D.CMDR / D.STSR	Floppy: command (write) / status (read)
2101H	D.TRKR	Floppy: track
2102H	D.SECR	Floppy: sector
2103H	D.DATR	Floppy: data (read/write)
2200H	H.KEY	Keyboard
2900H	CPDRA / PA.DTA / PA.DIR	IEEE-488 PIA: port A data / direction
2901H	CCRA / PA.CTL	IEEE-488 PIA: control A
2902H	CPDRB / PB.DTA / PB.DIR	IEEE-488 PIA: port B data / direction
2903H	CCRB / PB.CTL	IEEE-488 PIA: control B
2A00H	H.SCTRL (W) / H.SSTS (R)	6850 ACIA: control / status
2A01H	H.SXMT (W) / H.SREC (R)	6850 ACIA: transmit / receive
2C00H	H.VIO	Video memory controls (the ROM also uses +1, +2, +3)

occtxt6.ast names the same bases — H.FDC = 2100h, H.KEY = 2200h, H.IEEE = 2900h, H.SIO = 2A00h, H.VIO = 2C00h — which is worth noting because the CBIOS refers to the IEEE port as H.IEEE, not as a PIA.

Cross-check against docs/o1-memory-io-map.md §3

Map doc §3	Source	Verdict
2100–2103 floppy, 2100 stat/cmd, 2101 track, 2102 sector, 2103 data	D.CMDR/STSR/TRKR/SECR/DATR	**agrees**
2201–2280 keyboard, one address bit per row	H.KEY = 2200H, read with A0–A7 one-hot	**agrees**
2900–2903 IEEE-488 PIA, MC6821 (UC7)	CPDRA/CCRA/CPDRB/CCRB — a PIA with A/B data+direction+control, i.e. 6821-style	**agrees**
2A00–2A01 serial ACIA, MC6850 (UC4)	H.SCTRL/H.SSTS at 2A00, H.SXMT/H.SREC at 2A01	**agrees**
2C00–2C03 video PIA, MC6821 (UC15)	H.VIO = 2C00H, ROM touches +0…+3	**agrees**

No contradictions. The I/O map in `docs/o1-memory-io-map.md` §3 is now sourced from the firmware as well as the schematic and the running machine.

Free space within the window. The firmware touches exactly five bases (`2100`, `2200`, `2900`, `2A00`, `2C00`), so the clear gaps *between* the device blocks are `2300–28FF`, `2B00–2BFF` and `2D00–2FFF`. That is where a virtual peripheral belongs — but confirm the *decode granularity* against the schematic before relying on an address *inside* one of the used blocks, because several decode more than the registers the firmware names (the keyboard decodes A0–A7 one-hot, for instance). This narrows the recommendation in the map doc §7 and closes most of gap 1 in `docs/research-gaps.md`.

Monitor RAM and vectors

The ROM's scratch area sits immediately below video RAM, and the emulator, the video personality and any bus-level assertion need to know about it:

Address	Name	Note
0EA80H	MRAM	Base of the monitor RAM area (`ORG MRAM` in `occram*.asm`)
0EF00H	TEM	Used in the boot routines
0EF05H	RTRY	Retry counter
0EF08H	ROMRAM	ROM/RAM flag
0EF09H	DSTSB	Six bytes of disk info
0EF0FH	DMADR	Disk DMA address
0EF13H	SEKDEL	Disk step delay
0EF14H	SAVSEC / SAVTRK / SDISK	Saved sector / track / disk
0EF50H	HSTACT, UNASEC, LOGSEC	Host active, unallocated/logical sector
0EF59H	KEYLCK, CURS, LKEY, ESCH	Keyboard lock, cursor, last key, escape
0EF61H	PIAAD, PIABD	Saved PIA A/B data — the IEEE port's shadow
0EF6AH	DACTVE, BELCNT, LLIMIT	Disk active, bell counter, line limit
0EFF0H	INTBL	Interrupt vector table
0F000H	FWAVM	First address of video memory

Also from `occtxt6.ast`: `ROMVEC = 100H` (start of the ROM vector table), `NMIA = 66H` (NMI vector), `BKPI = 0CFH` (breakpoint interrupt = 1), `SEKTMO = 0FEH` (seek time-out status).

Bank switching: what the ROM actually does

The ROM defines exactly two bank-related macros:

```
ENADIM: MACRO          DISDIM: MACRO
        OUT 2                  OUT 3
        ENDM                   ENDM
```

`OUT 2` / `OUT 3` set and clear the `BIT 9` latch (the "dim bit", bank 3) — matching `docs/o1-memory-io-map.md` §4. There is **no `OUT 0` or `OUT 1` anywhere in the ROM**: the firmware never selects the RAM bank itself, so bank 1 ↔ bank 2 must be driven by hardware (reset and the NMI flip-flop chain), exactly as §4 describes. This is a useful negative result for the interposer: the firmware gives us no example of a software bank switch to model, and the "never drive `0x00`–`0x03`" rule in §4 stands.

Floppy command set

The WD179x commands the BIOS issues (`rom144.asm`, `;DISK EQUATES`):

Constant	Value	Command
`D.SEK`	`010H`	Seek
`D.STP`	`020H`	Step
`D.STPI`	`040H`	Step in
`D.STPO`	`060H`	Step out
`D.RDS`	`080H`	Read sector
`D.WRTS`	`0A0H`	Write sector
`D.RDA`	`0C0H`	Read address
`D.FINT`	`0D0H`	Force interrupt
`D.RDT`	`0E0H`	Read track
`D.WRTT`	`0F0H`	Write track

These are the type I/II/III/IV encodings of a WD179x-class part, which is what the floppy interposer (#20) has to speak. The MB8877-vs-WD1793 part question in `docs/o1-memory-io-map.md` §3 remains schematic-level and is unaffected.

IEEE-488 — and the parallel printer is the same port

The ROM carries a complete IEEE-488 implementation (`;IEEE-488 INTERFACE.`, line 2606, drivers from 3113), and it is the register-level contract for Drive C (#6/#22) and for the virtual printer (#19).

Three things worth extracting immediately:

The bus is inverted. Line 2668: *"ALL SIGNALS ARE LOW ON THE IEEE BUS WHEN PIA REGISTER CONTAINS '1'."* An emulated device that assumes positive logic will be wrong.

GPIB command bytes (`;IEEE control codes`):

Constant	Value	Meaning
IE_TALK	40H	make talker
IE_UTLK	5FH	make untalk
IE_LSTN	20H	make listener
IE_ULST	3FH	make unlisten

The PIA handshake programming (;IEEE EQUATES), including PA.CDT = 00101110b — *"to address port a data and set port a in input program handshake mode"* — plus the direction values (PA.DRO = 0FFH, PA.DRI = 00H, PB.DR = 0BFH, PB.DTO = 00000010b, PB.DTI = 00001011b) and the ready/strobe bits (PP.ORDY = 01000000b in port B, PP.IRDY = 10000000b in the PIA control register, STRB = 00100000b in port B). The current PIA state is shadowed in RAM at PIAAD/PIABD (0EF61H/0EF62H).

And one behaviour that matters more than the register values: line 3133 says *"IEEE always appears to be ready"* — the BIOS's device-status routine is effectively a stub. **The O1 will not poll a GPIB device for readiness**, so an emulated Drive C must be ready when addressed.

Finally, line 3249: *"The Parallel port is actually the IEEE port driven with the centronix [protocol]"*. The MX-80 personality (#19) is therefore an IEEE-488 device speaking Centronics on the same PIA, not a separate port.

Video, as the ROM drives it

- H.VIO is written as **pairs** — STO A,H.VIO+1 then STO A,H.VIO ("send data"), and STO A,H.VIO+3 then STO A,H.VIO+2 (lines 382–410).
- **Reading H.VIO clears the video interrupt** (line 2025, LD A,H.VIO ;clear interrupt) — the video PIA is one of the three IRQ sources at 60 Hz (docs/o1-memory-io-map.md §5).
- ENADIM / DISDIM toggle the dim bit through OUT 2 / OUT 3 (above).

Individual register semantics stay in docs/o1-memory-io-map.md §3; this pass records what the ROM does with them, not a re-derivation.

The 1.44 reassembly: the source *is* the shipped ROM (#203)

Measured 2026-10-13. rom144.asm assembles with the vendor's own assembler — SORCIM's ACT80 — to an image **identical, byte for byte, to research/roms/os1-144.rom (CRC32 c0596b14)**. That settles #83's deliverable 2: the source in this repository *is* the source of the shipped 1.44 monitor ROM, and the toolchain that produced it runs under MAME.

Assembler	`ACT80.COM` — SORCIM "Assembly Code Translator", **version 3.0**, safety copy in `research/act-assembler/`
Command	`ACT80 B:ROM144 H=B:` — the vendor's own, from `tran.txt`
Reported	`no ERROR`s, `473 Labels`, `Program LWA = 0FFFh`
Output	11 520 bytes of Intel HEX, 255 records, `ROM144.HEX` on drive B
Image	4096 bytes, CRC32 **`c0596b14`** — the shipped ROM
Reproduce	`tools/research/rom144-reassemble.sh` — the happy path, end to end

This is the happy path, and it is self-contained in the repository. The assembler is the vendor's own, the command is the vendor's own, and both live here: `research/act-assembler/ACT80.COM` and `tran.txt`. No substitution, no porting, no search for a missing version — the toolchain that built the part is the toolchain that rebuilds it.

ACT 3.0 and 3.2 — and no 3.5e

The archive holds nine copies of `ACT80.COM`, and they are **three distinct binaries over two versions**:

Version	Copies under …`/rlee/`	md5
3.0	`S/SORCIM/ACT Cross Assemblers/`, `M/MORROW/OUTSIDE SOFTWARE/ACT/`, `L/LOOSECPM/{079,109,262}/`	`4f76a911a66ddc8e1e02a1676c232d45`
3.2 (serial 279)	`L/LOOSECPM/{073,087,119}/`	`cf73590466736fd1960e829b5f2221aa`
3.2 (no serial line)	`D/DIGITAL RESEARCH/ECA/`	`88256731a9399dee68a29054865d072c`

`ACT 3.5e`, the version `bios141.gen` names, is not in the archive. The search was bounded rather than open-ended: every `ACT80.COM` under the archive tree found by name, hashed and string-searched for its banner, and the tree searched for an ACT `.DOC` or manual — the only hit was an unrelated accounts-receivable file. `READ.ME` in the kit lists the 3.0 files exactly as the archive holds them.

All three builds are now safety copies in this repository — `research/act-assembler/`, with the 3.0 kit's own `READ.ME`, `DOCOP80.ASM` (every ACT mnemonic, which is the file to diff against if a future source is rejected) and its `LINK` include files; the provenance rows are in `research/`

README.md, and the licence notice in the kit's READ.ME is recorded there rather than quietly dropped. The 3.2 builds sit beside it, so the byte-identical claim above can be re-checked against a local file rather than the mirror:

```
O1_ACT80=research/act-assembler/3.2/ACT80.COM tools/research/rom144-reassemble.sh
```

tools/research/rom144-reassemble.sh defaults to the 3.0 copy, so the build no longer needs the local archive to be present. The third build — the one that circulated inside a third-party bundle, 3.2/ACT80-eca-bundle.COM — is **not a drop-in**: it returns to the prompt printing nothing, where both SORCIM builds assemble cleanly. It is held as a curiosity, not as a spare.

The version gap turns out not to matter for this source. 3.2 assembles rom144.asm to a **byte-identical** hex file — same no ERRORs, same 473 labels, same LWA = 0FFFh, same 11 520 bytes. The only difference visible at all is one line of the summary, where 3.2 says 8230h bytes not used and 3.0 says 8232h; that is ACT's own bookkeeping, not the object code. So 3.0 is not merely "close enough" — for 1.44 it is exact, and the question of searching further for 3.5e can be closed as not-needed.

So the reassembly needs neither z80asm — the wrong tool, because this is an ACT job and not an 8080-syntax one — nor ACT 3.5e: the vendor's own assembler was in the local archive all along, and 3.0 and 3.2 both give the object code exactly (REQUIREMENTS.md).

ROM 1.3 is the same exercise one revision earlier, and the assembler is the one thing that did not carry over. The source disk reads — it is an 8-inch 77-track FM/MFM disk with CP/M skew 3, which tools/research/o1imd8.py handles — the source is extracted to research/source-code/ROM-13-source/, and the gap fill, the terminator and the CRLF handling all carried over from this build. But the 1.3 source uses the DBE/DCE pseudo-ops, which only ACT **3.5E** has, and the archive holds 3.0/3.2. The bytes those pseudo-ops emit are recovered from the shipped rev-B ROM on OCCROM.IMD, and a 1.3 image can be built with a mechanical source rewrite, but it is **not certified** — see docs/rom-13-reconstruction.md and #205. The project still has no use for a 1.3 ROM — we build from the 1.44 source — so this remains upstream's gain and nothing here depends on it.

The caveats in *Four things the emulated build teaches about the source*, and the damaged-record handling below, are the ones to carry into any other Osborne source assembled this way.

How the build is driven

The vendor's procedure (tran.txt) assumes two floppies: ACT80 in drive A, the source in drive B, ACT B:ROM144 H=B:. Reproduced under MAME with two DD images, and tools/research/rom144-reassemble.sh is the whole of it:

1. research/shipped-software/OS1SYSD.IMD ⟶ raw (imdinfo.py --raw), AUTOST.COM removed, ACT80.COM added ⟶ **drive A**. It is a 250 kbps MFM 40 × 5 × 1024 disk, and cpmtools' osborne1 diskdef is the right description of it: the CBIOS's own DPB agrees field for field (occbio95.asm, DPBGEN DSKD1,8*5,3,7,0,D1DSM,64,1100000000000000B,3 — 1 K blocks, AL0/AL1 marking the two directory blocks, offset 3 tracks).

2. The same disk emptied of files, with `rom144.asm` normalised to CRLF and written as `ROM144.ASM` ⟶ **drive B**. The source is 83 KB and the hex file is 12 KB; a full system disk has room for neither, which is the one thing that has to be arranged deliberately.
3. `o1raw2imd.py --dd` wraps both raw images (MAME will not open a bare sector image), `o1keys.sh` boots A with B in drive B and types the command, and `acthex.py` turns the hex back into an image and compares it against the shipped ROM.

Four things the emulated build teaches about the source

- **`rom144.asm` as captured is LF-only, with three stray CRs** (offsets 56 819, 59 483, 60 178 — lines 3198, 3334, 3383) where a line break lost its LF: `\tRET\r\tPAGE` is two lines, and so are the other two. CP/M text wants CRLF, so the *disk copy* is normalised CRLF ⟶ LF ⟶ CRLF, treating each stray CR as the break it was — **4658 lines rather than 4655**. The repository file is left exactly as captured.
- **The hex records do not cover the whole address space.** ACT emits a record only where the source put a byte, so `0x005C–0x0065`, `0x00E1–0x00FF` and `0x0FFF` are *gaps* — and the ROM holds **`00`** there, not `FF`. Filling the gaps with `FF` instead leaves 42 bytes wrong (`acthex.py --fill`). Worth knowing before the same exercise is run against ROM 1.3.
- **ACT's terminator is not Intel's.** The file ends with `:00010010` — a four-byte record carrying no data — rather than `:00000001FF`. Both 3.0 and 3.2 emit it identically, so it is ACT's.
- **The program is `ACT80.COM`, not `ACT.COM`.** The vendor's memo says to type `ACT`, which on this kit is CP/M's `<name>?` — command not found — and reads exactly like an ACT usage error. It is not one.

The one record that arrives damaged, and why it does not matter

On the floppy image, one record — `:100FC000…`, ROM addresses `0FC0–0FCF` — reaches the host with **six non-hex bytes in place of six characters** (`c0 de c0 de c0 0d`, at file offset 11 264: the first bytes of the file's last 1024-byte sector). The line is still 43 characters long, so nothing looks wrong at a glance. Three facts pin it down:

- **It is not the assembler's doing.** ACT 3.0 and ACT 3.2 are different binaries and produce byte-identical output, *including those six bytes*. Two assemblers do not invent the same garbage.
- **It is not a general fault in the floppy path.** A `PIP` copy of the 83 KB *source* through the same disk — guest write, MAME image save — came back byte-identical, so writing a file whose last sector is partial is fine.
- **The intended content is fixed by ACT's own bytes.** The record's checksum is intact, and it is exactly the value 16 bytes of `FF` at `0FC0` produce, as the neighbouring all-`FF` records confirm (`41` at `0FB0`, `21` at `0FD0`, `11` at `0FE0`). The record therefore says what it should have held, and the range is restored from that (`acthex.py --recover-ff`).

With the range restored, the assembled image is **identical to `research/roms/os1-144.rom` in all 4096 bytes** — and before the restore it already agreed in the other 4080. The cause of the six bytes is not isolated; it is deterministic, it lands in the file's last sector, and it is recorded here as an emulation observation rather than explained away.

What this closes, and what it does not

Gap (`docs/research-gaps.md`)	Status after this pass
1 — exact memory & I/O map, free decode windows	**Substantially closed.** Every address the firmware touches is now sourced, and the gaps between the five device blocks (`2300–28FF`, `2B00–2BFF`, `2D00–2FFF`) are clear. Decode *granularity* is still schematic-level.
3 — char-gen socket pinout	**Not addressed** — that is a hardware pinout, not firmware. Still open, still needed by #24.
10 — 6850 ACIA location for the WiFi modem	**Closed** — `2A00`/`2A01` confirmed by an independent source.
11 — floppy controller variant, port map, DD changes	**Port map and command set closed**; the DD changes are now documented (1.44 *is* the DD ROM, DD block = 1K, disk-type tables in the CBIOS). MB8877-vs-WD1793 remains open.

Two things this pass adds that no existing doc had:

- the **monitor RAM map** (`0EA80–0EFFF`) and the interrupt vector table at `0EFF0H`;
- the **GPIB control codes and PIA handshake constants** for the IEEE port.

Open questions / next steps

1. **Does the 1.44 ROM's CBIOS match the 1.41 source?** The CBIOS is disk-loaded, so extract it from a shipped system disk (`OS1SYSD.IMD` / `OS1SYSS.IMD`) and diff it against the 1.41 build.
2. **Read the IEEE driver section (lines 3113–3400) against `docs/drive-c-protocol.md`** to confirm the exact handshake the Drive C firmware expects (#6/#22).
3. **Pin down the video register semantics** from the ROM's write pairs (`+1`⟶`+0`, `+3`⟶`+2`) against the schematic (#24).

Provenance

- The source tree: `github.com/BrettHallen/Osborne_1` `ROM/Source_Code_v144/` (all 18 file sizes match). See `research/README.md`.
- The 1.44 binary used for the CRC comparison: `research/roms/os1-144.rom` (Don Maslin's archive) — identical to MAME's `3a10082-00rev-e.ud11`.
- The OCC engineering source disks (`research/source-code/*.IMD`, bitsavers) are the *disk* copies of the same material and have not yet been extracted — that is #2′s pipeline work.
- Licensing: `release.txt` retains Osborne's 1985 FOG letter verbatim; the terms are recorded in `research/README.md`.

ROM variants and third-party ROMs

Status: Solved as the ROM-identity authority. Every image this project holds is identified by **CRC32 over its raw bytes plus size**, cross-checked against MAME's driver, and `tools/`

`gen_rom_table.py` re-derives every CRC in the tables below from the bytes on every build — a document/bytes disagreement fails it. What is *not* settled is interpretation rather than identity: four stock revisions (BIOS A, 1.2, 1.2.1, 1.3) have never been dumped, so a machine fitted with one cannot be identified at all; one third-party label may be wrong; and one image's host-independence is unconfirmed. Those are the open questions at the end.

The answer. The machine has **two ROM sockets** — UD11, the 2732 monitor, and UA15, the 2716 character generator — and "the Osborne 1 ROM" is a *family*, because both were replaced or re-programmed by the upgrade vendors. What we hold is:

- **three dumped stock monitor revisions** — 1.4, 1.43 and 1.44 — plus a **rev-B image built by Osborne themselves** (`OCCROM0B.COM` on the OCC source disk, a vendor build artifact rather than a chip dump);
- **two character generators** that really are different typefaces — 93 of 128 glyphs differ — and five names for one of them;
- **third-party replacements**: OZROM 1E (IEEE-488 removed, video cut to 3 KiB), Nuevo 1.51, and the SWP co-processor's 8088 firmware; and an **8 KiB daughterboard image** holding OCC 1.44 and OZROM 1E with `A12` selecting;
- **four stock revisions we do not have**: BIOS A, 1.2, 1.2.1 and 1.3 — MAME carries them as `NO_DUMP`.

Those CRCs are what the personality gate in `firmware/rom-ident/` (#86) is built on, together with the byte-level properties in *Personality-relevant properties* below.

Issues: #70 (the inventory), #44 (the two character generators), #86 (the personality gate built on it), #72 (the ROM source), #205 (the 1.3 reconstruction, which is upstream's gain and not ours), #222 (which `NO_DUMP` slot the rev-B image belongs in — MAME's call), #74 (the two images whose source is still unrecorded).

> **Vocabulary:** `rev B` here is the **Osborne 1′s monitor ROM** (9/12/81) — a 1979-era ROM part, **not a board revision**. Board revisions belong to the boards; see [`hardware/README.md`](https://github.com/Myrcurial/o1next/blob/main/hardware/README.md).

The two sockets, and what a socket tap can read:

Socket	Package	Size	Role
UD11	2732 (24-pin)	4 KiB	Monitor / BIOS — the Z80 code that boots the machine
UA15	2716 (24-pin)	2 KiB	Character generator (the 6×9 font the video logic reads)

Method

Every image is identified by **CRC32 over the raw bytes** plus size. CRC32 is what MAME's ROM definitions and the archive indices both publish, so it is the only key that cross-references cleanly. Two independent sources agreeing on a CRC is treated as positive identification; a CRC agreeing under two different *names* is treated as a finding, not an error.

Stock monitor ROM (UD11, 4 KiB)

Rev	Our copy	CRC32	MAME name	MAME BIOS slot
A (1981)	—	—	`osba.bin`	`vera` — **NO_DUMP**
1.2	—	—	`osb12.bin`	`ver12` — **NO_DUMP**
1.2.1	—	—	`osb121.bin`	`ver121` — **NO_DUMP**
1.3	— (source extracted: `research/source-code/ROM-13-source/`)	—	`osb13.bin`	`ver13` — **NO_DUMP**; reconstructible in principle from the source disk `research/source-code/OSROM13.IMD`, blocked on ACT 3.5E — `docs/rom-13-reconstruction.md` (#205)
1.4	`tools/emulator-setup/roms/rev1.40.ud11`	`3d966335`	`rev1.40.ud11`	`ver14`
1.43	`research/roms/os1-143.rom`	`91a48e3c`	`rev1.43.ud11`	`ver143`
1.44	`research/roms/os1-144.rom`	`c0596b14`	`3a10082-00rev-e.ud11`	`ver144`

A rev-B image, from the OCC source disk (#205). `research/source-code/OCCROM.IMD` carries `OCCROM0B.COM` — a **vendor-built** 4096-byte monitor ROM, rev B, dated 9/12/81 — together with its source (`occrom0b.asm` plus nine modules) and its full listing (`OCCROM0B.PRN`). CRC32 **`f5f697c7`**, SHA1 `212b7992db91225909f47972799eeb878d9ed806`. It is **not a chip dump**: it is Osborne's own build output, shipped on their own source disk. It is consistent with both the source and the listing

on that disk, and it **boots** under `mame osborne1`, printing `OSBORNE SYSTEM ONE.M / REV B  227` and `INSERT DISK IN DRIVE ... AND PRESS RETURN.`

Its CRC is not one of the three known dumps, and rev B (9/12/81) sits between BIOS A and 1.3 — so it is a **candidate for the `osba.bin` / `osb12.bin` / `osb121.bin` `NO_DUMP` slots**, and *which* one is MAME's call, not ours. That question is filed upstream as #222. Two cautions: it is a vendor build artifact rather than a dump, and under MAME's `osborne1` video model the banner shows two stray `Ls` — rev B emits `ESC`+letter video escapes where 1.3/1.44 use single control bytes, and the model does not decode the older form (recorded, not explained). Extract it with `tools/research/o1imd8.py` + `cpmcp`; see `docs/rom-13-reconstruction.md` §6.1.

Corroboration: BrettHallen's `ROM/OCC1_v140.BIN`, `OCC1_v143.BIN` and `OCC1_v144.BIN` are byte-identical to the three we hold (same CRCs), and `research/source-code/BIOS-144-source/rom144.asm` is the original ACT80 source for 1.44 (#72).

Two facts about this table:

- **MAME's "ver144" is the part `3a10082-00` Rev E**, not a file named "1.44". The revision and the part number are two names for one image.
- **BIOS A, 1.2, 1.2.1 and 1.3 have never been dumped** (MAME carries them as `NO_DUMP` placeholders). We hold the **ROM 1.3 source disk**, so 1.3 is potentially reconstructible — but **not for us**: we build from the 1.44 source, which is in-repo and now proven to reassemble to the shipped part (#203). Reconstructing 1.3 would fill an *upstream* gap, so it is filed as a side quest, **#205**, and nothing in this project depends on it.

Character generator (UA15, 2 KiB)

There are **two distinct images** in circulation, and which one a machine has tracks the BIOS revision:

Image	CRC32	Paired with	Our copies
Early	`5297c109`	BIOS A … 1.4	`tools/emulator-setup/roms/char.ua15`
Late	`6c1eab0d`	BIOS 1.43, 1.44, and every ScreenPac / Nuevo machine	`research/roms/os1-vid8.rom`, `research/roms/os1vid80.rom`, `research/roms/OCC1_ScreenPac_RevA.BIN`

MAME's driver carries the early image five times, once per BIOS slot 0–4, with the comment *"this is CHRROM from v1.4 BIOS MB"*, and the late image for slots 5 and 6 (1.43 / 1.44). That is the evidence for the pairing, and it is the "early font / late font" distinction that #44 asks about — **both variants are captured in this repo.**

The socket itself (UA15, on mainboard sheet 4 of 9 — see `o1-mainboard-schematic.md` §4a) is wired **2732-style**: pin 21 is strapped to +5 V, and `/CE` (18) and `/OE` (20) are grounded. Both variants

are 2 KiB 2716s, so either drops straight in with no strapping change, and whichever is fitted is what a socket tap reads.

How the font is stored

The layout is **not** glyph-contiguous, and it is not obvious from the dump — it took MAME's driver to settle it (`src/mame/osborne/osborne1.cpp`):

```
u8 const gfx = m_p_chargen[(ra << 7) | (chr & 0x7f)];
```

so the byte for scanline `ra` of character `chr` is at `rom[ra * 128 + chr]`: all 128 characters' row 0 are contiguous, then all of row 1, and so on. Ten rows × 128 characters = **1280 bytes**, and the remaining 768 bytes of the 2 KiB are `0xFF`. Each row is one byte, **bit 7 = leftmost pixel**, and the cell is 8×10.

(Reading it as 128 glyphs of 16 bytes, or as 8 bytes per glyph, produces plausible-looking noise and no recognisable letters; MAME's driver had it in one line.)

That layout matters to the video personality (#24): a socket tap reads the font as `rom[ra * 128 + chr]`, and the Nuevo ROM adds an underline by forcing row 9 to `0xFF` when bit 7 of the character code is set.

The two fonts really are different — 93 glyphs of 128

Comparing the two images glyph by glyph (`tools/romfont.py` decodes both):

	Differing characters
0x00–0x1F graphics set	**0 of 32** — byte-identical
0x20–0x7E printable ASCII	**90 of 95**
0x7F and the rest	3

Only three printable characters are shared: space, the apostrophe and the backtick. So this is not a tweak — the late font is a different typeface, and the graphics set was carried over unchanged. That settles the open question below, and it means "which font is fitted" is a visible difference, not a subtle one.

Finding: one late image, five names

The bytes `6c1eab0d` appear under five different names across three sources:

Source	Name
Maslin's archive	`os1-vid8.rom`
Maslin's archive	`os1vid80.rom`
BrettHallen	`OCC1_char.BIN`
BrettHallen	`OCC1_ScreenPac_RevA.BIN` ("Screen Pac BIOS Rev. A")
MAME	`7a3007-00.ud15` and `character_generator_6-29-84.14`

So the stock late character generator, the ScreenPac's 2 KiB ROM and the Nuevo-era date-stamped character generator are **the same 2 KiB image**. The ScreenPac's 80/104-column capability therefore does not live in this ROM, and MAME models the ScreenPac as a *clone of* `osborne1` *with no ROM differences at all* (`osborne1sp` is a `COMP` of `osborne1`, while `osborne1nv` is a separate machine with its own ROMs). That matches the picture that the Nuevo 80-column implementation is a different design from the OCC one.

Third-party ROMs

ROM	Size	CRC32	Our copy	What it is
Micro Management **OZROM 1E**	4 KiB	`ba1f6131`	`research/roms/OZROM-1E/OZROM_1E.BIN`	Third-party replacement monitor ROM, © 1984 Micro Management. Strings: `OZROM 1E #00187`, `(c) 1984 Micro Management`, `Slip disk in drive and press RETURN.` Manual in the same folder.
Nuevo Electronics **Rev 1.51**	4 KiB	`298da402`	`tools/emulator-setup/roms/monrom-rev1.51-12.ud11`	Nuevo's replacement monitor ROM. Strings: `OSBORNE 1 COMPUTER, ROM Rev 1.51, COPYRIGHT 1983, OSBORNE COMPUTER CORP.`, `COPYRIGHT 1984, NUEVO ELECTRONICS CORP.` — i.e. an OCC ROM with Nuevo's DD/video patches. MAME name: `monrom-rev1.51-12.ud11`.
SWP co-processor	4 KiB	`948556db`	`research/roms/swp-p88.rom`	The SWP co-processor board's own ROM. **This is 8088 code, not Z80** (`8C C8 8E D8` = `MOV AX,CS` / `MOV DS,AX`), and it carries the WD1793 disk-BIOS error table (`Clock Error`, `Data CRC Error`, `Missing DAM`, `Lost Data`, `Seek Error`, `Write Protected`).
OCC 1.44 + OZROM combined	8 KiB	`edc31846`	*not copied*	Bret Hallen's 2764 image: the OZROM 1E at 0x0000–0x0FFF and the OCC 1.44 controller ROM at 0x1000–0x1FFF, in one dump.

Personality-relevant properties

Some interposer personalities are only compatible with some of these ROMs, so the firmware needs to know more than the CRC. The facts below are what the personality gate is built on (#86), each recorded with its evidence — the manual text or the bytes at the vector table.

ROM	IEEE-488 driver	Video window	Evidence
OCC 1.4, 1.43	**present**	4 KiB (`F000-FFFF`)	`013F-0156` holds eight `JP` vectors (`c3 36 09 …`), and the printer entry points `0187-0190` are `JP`s. Byte-identical between 1.4 and 1.43
OCC 1.44	**present**	4 KiB	Same structure with shifted targets (`c3 49 09 …`), and the implementation is in source we hold: `rom144.asm` `;IEEE-488 INTERFACE.` line 2606 (`o1-rom-source.md`)
OZROM 1E	**removed**	**3 KiB** (`F000-FBFF`)	Manual (`OZROM_1E_Manual.ocr.txt`), verbatim: *"The OZROM 1E does not support the IEEE-488 functions"*; *"013F-0156 WERE IEEE-488 ROUTINES / CALLING THESE FORMER JUMPS RESULTS IN A CONTROLLED SYSTEM CRASH (NO DAMAGE TO DISK DATA)"*; *"IEEE-488 AS IOBYTE DEVICE 3 NOW NULL DEVICE"*. Confirmed in the bytes: `013F-0156` is `ef f3 00 f7 …` — not jumps — and `0187-0190` is `f6 ff c9 f6 ff c9 af c9` `c9` = `OR 0FFH; RET` ×2 then `XOR A; RET; RET`, exactly the documented printer stubs. Video: *"only 3K of RAM is used for the video (F000-FBFF hex) since the last 1K is used for the redefinable keyboard and extra function keys*
OCC Nuevo + OZROM (8 KiB)	**present, unverified**	unknown	The sheet blocking interval (013F-0156) is eight bytes wide — so Nuevo kept the OCC routines, but nothing shows

Three consequences, all of which the gate encodes:

1. **Drive C (#6, #22, #84) and the virtual printer (#19) both need the IEEE-488 driver**, so both are refused under OZROM 1E. The printer is the less obvious of the two: the ROM's printer entry points `0187-0190` are precisely what OZROM stubs out.
2. **The video personality (#24) must cope with a 3 KiB window** under OZROM 1E, where `FC00-FFFF` belongs to OZROM's keyboard features.
3. **An unknown ROM must not be assumed good.** BIOS A/1.2/1.2.1/1.3 have never been dumped, so a machine fitted with one is unidentifiable — the gate refuses rather than guesses.

MAME cross-reference

MAME carries three Osborne 1 machines; the relationship between them is itself evidence about the upgrade boards:

MAME machine	Year	Parent	ROM set
`osborne1`	1981	—	BIOS A/1.2/1.2.1/1.3 (NO_DUMP) + 1.4 + 1.43 + 1.44, chargen early + late
`osborne1sp` "Osborne-1 with SCREEN-PAC"	1983	`osborne1`	**no ROM differences** — a clone, so the ScreenPac is modelled as hardware only
`osborne1nv` "Osborne-1 (Nuevo Video)"	1984	`osborne1`	`monrom-rev1.51-12.ud11` + `7a3007-00.ud15` + `character_generator_6-29-84.14` — its own BIOS and chargen

`tools/check_o1_map.sh` already selects a driver from the boot image's `52-`/`80-`/`104-` prefix (`osborne1` / `osborne1nv` / `osborne1sp`), so this table is also the map from video mode to ROM set.

Open questions, and the one that is closed

1. **Do the two character generators actually differ in the font? Yes, and substantially.** 93 of 128 glyphs differ, and the 32 graphics characters are byte-identical (see "The two fonts really are different" above). Only space, apostrophe and backtick are shared. `tools/romfont.py` decodes either image to a font, so the comparison can be repeated.
2. **What is BrettHallen's "Screen Pac BIOS Rev. A"?** It is byte-identical to the stock late character generator, so the label may be wrong — or the ScreenPac genuinely ships an unmodified char-gen and does its 80-column work entirely in hardware. Settle against the ScreenPac install and service pages (`research/screenpac/`, `research/osborne1/`). Affects #24.

3. **ROM 1.3: the source is extracted, the ROM is not certified — and the blocker is a missing assembler (#205).** We hold the 1.3 source disk (`research/source-code/OSROM13.IMD`, comment `rom 1.3, cbios .3 osborne source 12/16/81`) while MAME carries 1.3 as `NO_DUMP`. The disk is an **8-inch** 77-track FM/MFM disk with CP/M **skew 3**, and it has been read: the 19 files are in `research/source-code/ROM-13-source/`. The source uses the `DBE`/`DCE` pseudo-ops, which only **ACT 3.5E** has — every `ACT80.COM` in the local archive is 3.0/3.2 — so a byte-exact reconstruction needs an assembler we do not have. What those pseudo-ops emit *is* recovered (from the shipped rev-B ROM on `OCCROM.IMD`), and a 1.3 image can be built with a mechanical source rewrite, but it is **not certified** and must not be presented as the real ROM. Full account, including the control build that decides it: `docs/rom-13-reconstruction.md`. **The project itself has no use for a 1.3 ROM** — everything we build is built from the 1.44 source — so this is upstream's gain, not ours, and nothing here depends on it.
4. **Confirm `swp-p88.rom` is host-independent.** The working assumption is that the co-processor board and interposer are common and only the host driver differs, which makes this the CoPower-88 ROM rather than a Kaypro-only artefact. A disassembly would confirm it and simultaneously yield the host mailbox protocol — #3/#4/#5.

Provenance

- Stock monitor ROMs `os1-143.rom`, `os1-144.rom`, and the character generators `os1-vid8.rom` / `os1vid80.rom` — Don Maslin's ROM archive on retroarchive.org.
- `OZROM-1E/*`, `OCC1_ScreenPac_RevA.BIN` — github.com/BrettHallen/Osborne_1 `ROM/` (the OZROM manual is his cleaned-up copy).
- `tools/emulator-setup/roms/*` — the MAME `osborne1` / `osborne1nv` ROM sets, matching the CRCs MAME's driver publishes.
- ROM 1.44 source — `github.com/BrettHallen/Osborne_1` `ROM/Source_Code_v144/`; the OCC engineering source disks are from bitsavers.
- Cross-reference data — MAME `src/mame/osborne/osborne1.cpp` (`ROM_START( osborne1 )`, `ROM_START( osborne1nv )`, and the `COMP()` lines).

See `research/README.md` for the per-file archive provenance, and #74 for the two images whose source is still unrecorded.

ROM 1.3: the source disk, and what is left

Status: Partially solved. The disk is read and the source is **extracted, indexed and readable**. What is *verified*: the `DBE`/`DCE` rule, and a rev-B control build that reproduces a shipped Osborne ROM to 3759 of its 4096 bytes. What is **not**: the 1.3 image itself, which is reconstructible but **not certified**, and must not be handed upstream as the real ROM. Two things stand in the way and both are *tooling*, not knowledge: the 1.3 source uses the `DBE`/`DCE` pseudo-ops, which only ACT **3.5E** has, and every build loses 337 bytes of its hex output to an emulation defect (§7). Everything needed to finish it is in this repo or named below, and the recommendation is to record the result and stop (§10). **Issues:** #205, #72, #70, #203.

> **Vocabulary:** `rev B` here is the **Osborne 1′s monitor ROM** (9/12/81) — a 1979-era ROM part, **not a board revision**. Board revisions belong to the boards; see [`hardware/README.md`](https://github.com/Myrcurial/o1next/blob/main/hardware/README.md).

MAME carries BIOS A, 1.2, 1.2.1 and 1.3 as `NO_DUMP` placeholders, so `mame osborne1 -bios ver13` cannot run. We hold the 1.3 **source disk** — `research/source-code/OSROM13.IMD`, whose own comment reads `rom 1.3, cbios .3  osborne source 12/16/81` — and this is what came of trying to turn it into the missing `osb13.bin`.

The answer, up front

1. **The disk reads, and the source is complete.** `OSROM13.IMD` is an **8-inch** disk, not an Osborne 1 5.25" one, which is why no earlier reader in the tree could describe it. It yields 19 files — `OCCROM0..9.ASM` (the monitor ROM, split into modules) and `OCCBIO0..8.ASM` (the CBIOS) — all of them clean text that starts with a `TITLE` and ends with `; END OCCROMn?.???`. They are committed at `research/source-code/ROM-13-source/`.
2. **The build needs ACT 3.5E, and we do not have it.** The source uses two pseudo-ops, `DBE` and `DCE`, that exist in no ACT the archive holds — every `ACT80.COM` in the local archive is 3.0 or 3.2 (or ACT II 8086 3.4), and none of them carries those strings in its mnemonic table. The rev-B listing on `OCCROM.IMD` names the version it *was* built with: `SORCIM 808x Assembler ver 3.5E`.
3. **What `DBE`/`DCE` do is recovered exactly**, from a shipped ROM we *do* have: the rev-B disk carries `OCCROM0B.COM` (4096 B) and its full listing, and comparing the two gives one rule holding over every line. See below.
4. **A 1.3 image can be produced** — `tools/research/rom13-reassemble.sh` does it, by rewriting `DBE`/`DCE` into `DB` so ACT 3.0 emits the same bytes — but **it is not certified**: three errors in the 1.3 build are still unidentified, and 337 bytes of every build's hex output are lost before they reach the disk. It must not be handed upstream as the real 1.3 until the control build passes. §6.
5. **The hex output arrives damaged at about six fixed *file offsets* per build** — identically on a double-density disk and on a single-density one, and on drive A or drive B — so it is not the disk layer. §7.
6. **Recommendation: stop here, as "record it and stop".** The source work is done and committed; what is left is two *tooling* defects (§6, §7) whose clean fix is the assembler we cannot get. Nothing in the project needs a 1.3 ROM, and the durable half of the result — the disk read, the source, the recovered `DBE`/`DCE` semantics, and a control build that reproduces **3759 of a shipped Osborne ROM's 4096 bytes exactly** — is already banked. §10.

1. What the disk is

`imdinfo.py` reports the geometry; `tools/research/o1imd8.py` reads it.

Cylinders	77, one head, 630 784 bytes
Track 0	250 kbit/s **FM**, 26 sectors of **128** bytes (the IBM-3740 boot track)
Tracks 1–76	250 kbit/s **MFM**, 8 sectors of **1024** bytes
Filesystem	CP/M 2.2, 2 KiB blocks, **128** directory entries (4096 B = two blocks), two reserved tracks
Skew	**3** — the sectors of a track are interleaved

Seven disks in `research/source-code/` share this format (`OCCCBIOS`, `OCCROM`, `OCCUTIL`, `OSBIOSSR`, `OSBROM`, `OSROM13`, `OSUTLSRC`), so the reader is a tool, not a one-off.

The skew is the whole difficulty

`o1imd8.py` writes the image in the linear shape a `cpmtools` diskdef expects (track 0's 3328 bytes padded out to 8192, `boottrk 2`), and the diskdef carries `skew 3`. Without the skew the read **still works** — the directory parses and every file comes back — from the wrong sectors. That is the failure mode `docs/ozrom-1e.md` §10.4 warns about, and it is how this one was found: content chaining. At skew 3 every sector boundary in `occrom33.asm` joins mid-token to the next —

```
BDOS (f     + unction  ->  BDOS (function
\tJMP\t     + LIST      ->  \tJMP\tLIST
;HOUSE      + KEEPING   ->  ;HOUSEKEEPING
\tLD\tA,PIAB + D        ->  \tLD\tA,PIABD
```

— and at skew 1 the joins are `;continueT`, `indicate RY` and a sector that should not follow. `skew 3` is also exactly `tools/cpmfs.py`'s `skew_table(3, 8) = 0 3 6 1 4 7 2 5`, i.e. `cpmtools`' own table (`cpmfs.c`), which is the cross-check that the two agree.

2. The source set, and what is missing from it

`OCCROM03.ASM` is the master. It sets `ORG 0`, defines `ALPUSH`/`ALLPOP`, and `LINK`s the rest:

```
    LINK    OCCROM13.ASM    ;ROM STANDARDS
    LINK    OCCROM23.ASM    ;CPM BOOT
    LINK    OCCROM33.ASM    ;CONSOL ROUTINES
    LINK    OCCROM43.ASM    ;PIA ROUTINES
    LINK    OCCROM53.ASM    ;SIA ROUTINES
    LINK    OCCROM63.ASM    ;DISK ROUTINES
    LINK    OCCROM73.ASM    ;DIAGNOSIC ROUTINES
    LINK    OCCROM83.ASM    ;FORMAT ROUTINE
    LINK    OCCROM93.ASM    ;UTILITES
    LINK    OCCRAM2.ASM ;RAM LOCATIONS
```

File	Size	On OSROM13.IMD?
occrom03.asm … occrom93.asm (10)	896 … 32 512 B	yes
occbio03.asm … occbio83.asm (9)	640 … 16 896 B	yes
occram2.asm	3840 B	**no** — recovered from OSUTLSRC.IMD
occtxt6.ast	9472 B	**no** — recovered from OSUTLSRC.IMD

Two files the build needs are not on the ROM disk at all. OCCRAM2.ASM is the ;RAM LOCATIONS module (ORG MRAMEXT, ORG MRAM); the utilities-source disk OSUTLSRC.IMD carries it. The systext (OCCTXT) is the equates file the ROM reads for CR, LF, MTRK, EOFC, MRAM, MRAMEXT, KL_LEN, ALVS, CSVS and the rest; **no OCCTXT is on the 1.3 disk either**, and the one committed here is occtxt6.ast from OSUTLSRC.IMD — the nearest revision we hold to 12/16/81. That is an *assumption*, and it is the one place a future find could change the bytes: occtxt5.ast and occtxt6.ast are identical apart from the header and BIOS, and BIOS is referenced by no ROM source line, so the choice is inert *for this source* — but a later OCCTXT is not guaranteed to be.

3. Obstacle one: CCP is defined twice

The vendor's recipe for the CBIOS is ACT OCCBIO05.ASM S=OCCTXT6.AST R=F L=A: (bios141.gen), so S= is how the systext is fed in. Feeding occtxt6.ast to the ROM master fails immediately and *fatally*:

```
 *Error page   0        SORCIM 808x Assembler ver  3.0   00/00/00 00:00  Page   1
  Osborne Computer Corp System Text.                              A:OCCTXT6 .AST
D       = D000      CCP =   BIOS-1600h  ;CCP entry
   1 ERRORs,  716 Labels, 75A7h bytes not used. Program LWA = 0000h.
```

The master sets CCP = 0 ("Developed for use with the new cbios") and every OCCTXT we hold defines CCP = BIOS-1600h. ACT 3.0 treats the duplicate as fatal and abandons the assembly. The 1.44-era systext (SVER = 14) has *removed* MSIZE/BIOS/BDOS/CCP altogether, which is evidence that this line was on its way out; the revision contemporary with 12/16/81 is not in the archive.

rom13-reassemble.sh therefore disables that one line in its *copy* of the systext, leaving the source alone. **CCP is not referenced by any ROM source line** (it appears only in the master's own definition and in comments), so the change cannot move a byte of object code — and the rev-B control build, whose master does *not* define CCP, needs no such change and passes the same way. The edit lives in the script rather than in a source file, and the tooling hazards that surround the build are in §7.

4. Obstacle two: DBE/DCE are ACT 3.5E only

Assembling without a systext gives **241 errors**, all U (undefined symbol) on things like CMP MTRK, CPI EOFC, LK C,CR, LDK HL,D.TRKR — the systext fixes those. With the systext and the CCP line disabled the count drops to **27**, and every one of them is on a DBE or DCE line:

```
I 0E3B  000000      :xmsg:  DBE     cr,lf,'Program begins'   ! DCE ':'
```

Neither mnemonic exists in ACT 3.0 or 3.2 — not in `DOCOP80.ASM` (which lists all 130 ACT mnemonics) and not in the mnemonic table inside `ACT80.COM`, which is plain text: `… DA DB DCR DCX DC DEC DI DJNZ DS DW …`, with nothing between `DB` and `DCR`.

What they emit, recovered from a shipped ROM

`OCCROM.IMD` (the rev-B ROM source, 9/12/81) is the disk that makes this answerable: it carries the *source* (which uses `DBE`/`DCE`), the **shipped image** `OCCROM0B.COM` (4096 B), and the **full listing** `OCCROM0B.PRN` — so every DBE/DCE line can be read against the bytes it produced. The listing gives the address of each line and the ROM gives the bytes:

```
0D3A  34          DIAMSG: DBE 'Z'-40h
0D3B  404040A49E      DBE '   ROM DIAGNOSTICs'
0D4D  1A14A6CAD8      DBE cr,lf,'Select Test:'
0D9E  A6CAD8CAC6      DCE 'Select (A,B): '
```

Decoding those (and the other eleven lines) gives one rule, with no exceptions:

```
byte = value * 2          DBE — every byte even
byte = value * 2 + 1      DCE — the *last* byte only
```

`0D3B`: `' '`=0x20⟶0x40 ×3, `'R'`=0x52⟶0xA4, `'O'`=0x4F⟶0x9E, … `'s'`=0x73⟶0xE6. `0D4D`: `cr`=0x0D⟶0x1A, `lf`=0x0A⟶0x14, `'S'`=0x53⟶0xA6, … `':'`=0x3A⟶0x74. `0D9E` (a `DCE`) is the same shift, and its **last** byte is `0x41` = 0x20×2+1 — the end-of-string marker. The rev-B ROM contains 18 literal `0x7F` bytes, but none of them falls inside a DBE/DCE string, so the "`7F`, repeat count, char" escape `rom144.asm` documents is not exercised here.

`tools/research/actdbe.py` rewrites each `DBE`/`DCE` statement into `DB` statements whose operands are the same expressions multiplied by two, leaving ACT to evaluate `cr`, `lf` and the rest itself, and keeping the original statement as a trailing comment so the result is auditable. Two things the translation has to get right, and both fail silently: the CP/M source is **CRLF** (emit bare LF and the source text lands in the hex file as object code), and `!` is ACT's **statement separator** while ACT's local labels are `:name:` with a leading colon.

5. What is verified, and what is not

Claim	Evidence
The disk reads and the source is complete	19 files, sizes matching the directory, all clean text, all `; END OCCROMn?.???` terminators
The format is skew 3	every sector boundary in `occrom33.asm` chains mid-token at skew 3 and nowhere else; matches `cpmfs.py`/`cpmtools`' own table
`DBE` = value×2, `DCE` = value×2 with the last byte +1	15/15 lines in the rev-B listing against the shipped `OCCROM0B.COM`
The translated rev-B build reproduces the shipped rev-B ROM, as far as its output survives	`no ERRORs`; `LENGTH OF THIS ROM IS = 0FC6`, the shipped listing's own number; and **0 differing records** against the shipped hex — **3759 of the shipped image's 4096 bytes byte-identical**. The other 337 bytes are the 16 records lost to §7, so they are *unverified*, not verified
The translated 1.3 build is **the ROM**	**NO** — 3 errors remain and 337 bytes are lost to §7; see §6
The built 1.3 image boots	**NO — run, and it fails.** MAME renders it as a full screen of block glyphs, no banner. Method and screenshot: §6.1
The vendor-built rev-B image boots	**YES** — `OCCROM0B.COM` (CRC32 `f5f697c7`) prints `OSBORNE SYSTEM ONE.M / REV B  227` and `INSERT DISK IN DRIVE ... AND PRESS RETURN.` under MAME. §6.1

6. The control build, and what still stands in the way

The check that matters is the rev-B control build: build rev B from `OCCROM.IMD` with the same translated source and the same ACT 3.0, and compare against the `OCCROM0B.COM` that shipped on that very disk.

The assembler half of that passes. One `actdbe.py` defect had to be fixed first: an unmatched optional group in the label regex emitted the literal text `None` as a label on every unlabelled `DBE`/`DCE` statement, so ACT defined a symbol called `None` over and over — 25 errors in the 1.3 build, and in the rev-B build an `LWA` of `0FAAh` against the shipped listing's `0FC6h`, with labels late in the ROM resolving 28 bytes low. With the regex fixed (`label or ''`), rev B gives:

```
       'LENGTH OF THIS ROM IS = 0FC6'
 no ERRORs,  696 Labels, 75D7h bytes not used.     Program LWA = EFEFh.
```

`0FC6` is the shipped listing's own number, and **every record the build emits is byte-identical to the shipped hex** — 0 differing records, checked record by record. That is **3759 of the shipped**

image's 4096 bytes verified identical, and the 337 bytes that are not are the 16 records §7 loses: unverified, not verified. So ACT 3.0 plus `actdbe.py` reproduces a shipped Osborne ROM *as far as its output survives*, and the `DBE`/`DCE` recovery is sound.

Why that is checkable for rev B and not for 1.3 is the whole point. The rev-B disk carries the *answer*: `OCCROM0B.COM`, 4096 bytes, CRC32 `f5f697c7`. So the damage shows up as a discrepancy against a reference, and what it ate can be read off that reference. Our own build of the same source is CRC32 `5f48a26a`, and the 337 bytes between the two are exactly §7′s damage — nothing else differs. **For 1.3 there is no reference.** The same defect eats the same kind of bytes and nothing can tell us what they were.

So the toolchain is *verified* for rev B and only *trusted* for 1.3, and that difference is not a property of the toolchain — it is a property of whether the answer already exists. A control build is a check, not a generator: it can tell you that a build is right, and it cannot supply the bytes a build failed to produce.

There is a second, independent blocker: **the 1.3 build carries three errors that the rev-B build of the same code does not.** Even with a clean output path the 1.3 image would not be right until those are found.

6.1 The boot test, which the issue makes the pass/fail

It runs, and it says no. Two things had to be learned to run it at all:

- **`o1keys.sh` segfaults whenever `-bios` is passed** — with *any* ROM, including the stock `rev1.40.ud11`. MAME itself runs fine with `-bios` (`exit 0`), so it is the harness's VRAM reader, not the guest. The way round it is the one `docs/mame-emulation.md` §4 already uses: put the candidate ROM in as the **default** BIOS file (`3a10082-00rev-e.ud11`) in a scratch rompath and pass no `-bios` at all. MAME prints `WRONG CHECKSUMS` and continues.
- **A screenshot is the only readable output**, since the harness cannot be used. A four-line Lua script — `emu.register_frame_done`, then `screen:snapshot()` — writes a PNG that can be looked at.

With that, booting a scratch rompath whose default BIOS is **the vendor-built rev-B image** (`OCCROM0B.COM`, CRC32 `f5f697c7`, extracted from `OCCROM.IMD`) gives its own sign-on and prompt:

```
LOSBORNE SYSTEM ONE.M
    REV B   227

INSERT DISK IN DRIVE LAM AND PRESS RETURN.
```

That is a boot, and the banner matches the rev-B source's own `;VER = REV B` / `;DATE = 227`. (The stray `Ls` are the ROM's `DB ESC,'L'` / `DB ESC,'M'` video escapes printing literally; rev B escapes with `ESC`+letter where 1.3 and 1.44 use single control bytes, and MAME's `osborne1` video model does not decode the older form. Unexplained beyond that, and it is recorded rather than guessed at.)

Booting **our built 1.3 image** the same way gives a full screen of block glyphs and no banner. **So the 1.3 image fails its own pass/fail** — the check ##205 asks for, and it fails it — and the failure is consistent with the two defects above rather than with anything about the source.

What stops the comparison passing outright is §7: the emulated write-back damages six hex records per build, and the damage is identical on every run, so neither `--recover-ff` nor a union of runs can put them back. That is now the *only* thing between us and a certified rev-B reproduction — and therefore between us and a certified 1.3 image.

The 1.3 build carries three further errors that are not yet identified. It is not the translation (the rev-B build has none), it is not `.siz` (ACT 3.0 handles the `occtxta.ast` that has it, and `occtxt6.ast` does not), and it is not ~ (NOT) or the `RRC`/`DAA`/`ADC A,40H` instructions — all three were tested directly. Its `LENGTH` comes out `0FFA`, inside the 4 KB window, so no overflow error is firing.

So: a 1.3 image can be built, and `rom13-reassemble.sh` builds one, but it is not the shipped 1.3 ROM and must not be presented as one.

ACT 3.5E is the clean answer and it is unobtainable — the maintainer has said so, and it is recorded here so nobody goes looking again. It is in no local archive (nine `ACT80.COM` copies checked, plus the schorn.ch `act.dsk`) and not on bitsavers. That matters because it means the two defects below have no clean fix available, which is the whole basis of the recommendation in §10.

What is left, in order of how much it would buy:

1. **The output damage in §7** — locate it and the rev-B control build passes, and the 1.3 image falls out of the same toolchain.
2. **The three 1.3 errors** — a source-level difference from rev B, still unidentified; the rev-B build has none.
3. The **`OCCTXT` revision contemporary with 12/16/81**, in case the equates differ from `occtxt6.ast` (the choice is inert for this source, but a later revision is not guaranteed to be).
4. The **`OCCRAM2.ASM` from the 1.3 era** — the copy used here is from 10/9/81.

7. A tooling hazard: MAME's floppy write-back

The assembler's hex output arrives on the floppy image with **runs of `0xDE 0xC0`** where hex digits should be — about six records per build, each one truncated mid-record with the next record's text running on behind it:

`:1001C00\xde\xc0\xde\xc0\xde\xc0…:1001D00…`

`acthex.py` reports them as `not a usable record`; a clean build has none. Five measurements narrow it down, and the last two are what close the question:

- the *source* files and `ACT80.COM` read back off the same floppy **byte identical** to what was written, so the disk is not generally corrupt, and the guest's *read* path is sound;
- **three runs of the same build are byte-identical files** (`md5` equal) — deterministic, not flaky, so a union of runs recovers nothing;

- the damaged bytes land at the same **output-file offsets** in both builds (0x480 in each) while the *object addresses* underneath them differ (0x1C0 for rev B, 0x1F0 for 1.3) — so it follows the output stream, not the object;
- every damaged run starts on a **128-byte boundary inside its sector**, i.e. a CP/M record boundary;
- it is **identical on a single-density disk (256-byte sectors) and on a double-density one (1024-byte sectors)**, and on drive A or drive B — so it is not the disk format, the sector size, or the image layout.

That leaves the output path *above* the disk layer: the bytes are already wrong when they are handed to the emulated controller. It is the same class as ##203's one damaged record — `acthex.py`'s `--recover-ff` exists for the shape seen there — but this is six records, and their content is not FF, so that recovery does not apply.

7.1 What the ROM 2.0 build measured about the same damage

The ROM 2.0 build (#247) hit this again, in the `C0 DE` shape `acthex.py` names, and those measurements narrow it further: they rule out the disk, the drive and the host's speed, and leave the *build's own byte stream*:

- the damaged bytes are in the **raw image the emulator wrote back**, not in the read path: `imdinfo.py` fills an *unavailable* sector with `E5`, and these runs are `C0 DE`, so the image already held them (measured on `run_b.raw`);
- **sending the hex to the other drive changes nothing.** The same record was damaged with `ACT80 B:ROM2 H=B:` and with `ACT80 B:ROM2 H=A:` — a different disk, a different allocation, the same `C0 DE` at the same record;
- **throttling changes nothing.** Five `-nothrottle` runs and one run at the machine's real speed (`O1_MAME_EXTRA=`) produced byte-identical damage — the same result §7′s five measurements give for the 1.3 build;
- what *did* change it was the source. The preceding build of the same code — with two bytes of inline prose in `rom2.diff` that emit nothing — had **no** damaged records at all, and moving that prose out of the diff's code lines (so the assembled `ROM2.ASM` went from 90676 bytes to 90607, both inside the same 177 CP/M blocks) brought the clean build back. The damaged record is the same one either way, with the same bytes under it.

So the variable this build can control is the size and content of what the guest writes, not where it writes it. What that means in practice is recorded where it bites: `rom2asm.sh`'s step 8 refuses the build rather than padding the hole, and `rom2blob.py` refuses it a second time — so the next person to hit this gets a failure, not a silent artifact. `acthex.py --merge` and `rom13-reassemble.sh`'s `O1_RUNS` remain the union-of-runs answer, and they only help once the damage moves between runs, which for this shape it did not.

8. What upstream gets, and what it should be told

MAME wants `osb13.bin`, 4096 bytes, inside `osborne1.zip`. We can hand over the *source*, the *method*, and the two things we cannot supply. The honest note to the MAME devs is:

Osborne 1 ROM 1.3 is not a dump — we do not have a 1.3 machine. It is reconstructed from Osborne's own source, which survives on the OCC engineering source disk `OSROM13.IMD` (bitsavers; also in the o1next repo), read as an 8-inch 77-track FM/MFM disk with CP/M skew 3. The source is 10 modules (`OCCROM0..9.ASM`) linked by `OCCROM03.ASM`, plus `OCCRAM2.ASM` and the `OCCTXT` systext from `OSUTLSRC.IMD`. **The source needs SORCIM ACT 3.5E** — it uses the `DBE`/`DCE` pseudo-ops, which 3.0 and 3.2 do not have, and the shipped rev-B ROM on `OCCROM.IMD` was itself built with `ver 3.5E`. We recovered what those two pseudo-ops emit (each byte is the value shifted left one; `DCE` sets bit 0 of the last byte) from that disk's own `OCCROM0B.COM`, and can build a 1.3 image with ACT 3.0 and a mechanical source rewrite. The toolchain checks out: a control build of the rev-B ROM from that same disk gives no errors and reproduces 3759 of the shipped image's 4096 bytes byte for byte — the other 337 are lost to a defect in the emulation we build under, which damages a handful of hex records on the way back out. So the 1.3 image is **not certified**, and it should not go into a romset as if it were a dump.

So: this is a *reconstruction recipe*, not a dump. If you have ACT 3.5E the rewrite is unnecessary and it finishes in one run — **we have stopped looking for it ourselves**, and we will update this note if it ever turns up.

The point of handing that over rather than a hopeful 4 KB blob is that a wrong `osb13.bin` in a romset is worse than a placeholder: it would be indistinguishable from a dump afterwards.

9. Reproducing this

```
# the source, read out of the disk (needs cpmtools)
python3 tools/research/o1imd8.py research/source-code/OSROM13.IMD /tmp/rom13.raw
cd /tmp && cpmls -T raw -f occ8 rom13.raw          # 19 files

# the whole build, plus the rev-B control, under MAME
tools/research/rom13-reassemble.sh --validate /tmp/o1-rom13

# what a build reports
grep -E "ERROR|LWA|not used" /tmp/o1-rom13/act.log
```

`rom13-reassemble.sh` writes `/tmp/o1-rom13/OCCROM03.HEX` (ACT's own output), `os1-13.rom` (the 4096-byte image), and — with `--validate` — `revb.bin` for comparison against the shipped `OCCROM0B.COM`. It exits non-zero when the control build does not match, which is the honest state today.

10. What this closes, and what it does not

	Status
`research/source-code/*.IMD` are readable	**Closed** — `tools/research/o1imd8.py`; all seven disks, not just 1.3
The 1.3 source is extracted and indexed	**Closed** — `research/source-code/ROM-13-source/`
`DBE`/`DCE` semantics	**Closed** — recovered and verified against the shipped rev-B ROM
MAME's `osb13.bin`	**Open, and recommended to stay so** — see below
The other `NO_DUMP` slots (A, 1.2, 1.2.1)	**Open, and closer than they were** — `OSBROM.IMD`, `OCCROM.IMD` and `OSBIOSSR.IMD` hold the rev A/rev B source sets and `OCCROM.IMD` even carries a *shipped* rev-B image (`OCCROM0B.COM`, 4096 B). Whether that image is one of MAME's missing slots is a separate question and is not answered here.

Recommendation: stop here

We do not have a 1.3 ROM, and the side quest should stop at this point — "record it and stop", which is what #205 allows for.

The reason is not that the source is short of anything. It is that the two things still in the way are **tooling defects, not knowledge**, and the clean fix for both is an assembler that is unobtainable:

- the hex output is damaged before it reaches the disk (§7), and
- three errors in the 1.3 build are unidentified while the rev-B build of the same code has none (§6).

Against that, the durable half is already banked: the disk reads, the source is committed, the `DBE`/`DCE` semantics are recovered and verified, the control build reproduces 3759 of a **shipped** Osborne ROM's 4096 bytes exactly, and the recipe is written down. None of that needs a 1.3 ROM to be useful, and no project path does either — we build from the 1.44 source.

What would justify picking it up again: a clean output path (a fixed emulation, or any environment where ACT's hex file survives), or the `OCCTXT`/`OCCRAM2` of the right vintage. Both are cheap to test and both are written down here. Until then, more hours buy nothing, and a *wrong* `osb13.bin` in a romset would be worse than the placeholder it replaced.

Related: #72 (the ROM source index), #70 (ROM inventory), #203 (the 1.44 reassembly and the ACT80 safety copy), `docs/o1-rom-variants.md`.

Reading the monitor ROM off a real machine

Status: Partially solved. The *identification* is settled and needs no tool: the upper line of the sign-on is framed by the underline escape and survives a faulty dim bit, so a machine's ROM revision is readable off the screen (§1), and `^D` at the boot prompt separates the early ROMs from the 1.40-and-later ones (§1a). The *copy* is written down to the byte (§2, §4) but has not been run end to end, and the machine this was written for is out of play (§7a), so there is at present no hardware route to any `NO_DUMP` slot. **Issues:** #205, #70.

MAME carries BIOS A, 1.2, 1.2.1 and 1.3 as `NO_DUMP`, and the one thing that would settle 1.3 is a **dump from a machine that has one**. This is the procedure for reading it off a real Osborne 1 — what can be done from inside the machine, what each route needs, and what the bytes should look like so the answer is unambiguous.

Nothing here writes to a diskette. Every route below is read-only on the media.

> **Vocabulary:** `rev B` in this document is the **Osborne 1's monitor ROM** (9/12/81, the `OCCROM0B` lineage) — it is **not a board revision**. Board revisions in this project belong to the boards; the register and the naming rule are in [`hardware/README.md`](https://github.com/Myrcurial/o1next/blob/main/hardware/README.md).

1. First, the free answer: read the *upper* line of the sign-on

The sign-on has two lines and **they are framed by different video escapes** — the upper one by `ESC 'l'` … `ESC 'm'`, the lower one by `ESC ')'` … `ESC '('`. They are the **underline** pair and the **half-intensity** pair respectively, and that is why a machine with a faulty dim bit loses exactly the revision line and nothing else. Confirmed on a real machine: the upper line renders **underlined**, and the revision line is the one that goes.

The two attributes live in different places, which is the whole reason the fault is so selective. Underline is **bit 7 of the main video RAM**; dim is **bit 8 — the dedicated 4116 in bank 3** (`docs/o1-memory-io-map.md` §2). So a fault in the bank-3 attribute path (that 4116, or the `BIT 9` latch that maps it) costs the dim line and leaves the underlined one perfectly legible. One chip, not the video RAM.

So: **the upper line is readable even when the version line is not, and the upper line differs by revision.**

Upper line reads	ROM
`OSBORNE SYSTEM ONE.` (all caps)	**rev B**, 9/12/81
`Osborne System ONE.` (mixed case)	**1.3**, 12/16/81
`OSBORNE 1`	1.4 / 1.43 / 1.44 and later

The lower line, for completeness: `MONITOR 1.3` on 1.3, `Rev 1.44 (c) 1983 OCC` on 1.44, `REV B 227` on rev B.

This is derived from the sources, not from a guess: `occrom1b.asm` line 338 is `DB 'OSBORNE SYSTEM ONE.'`; the 1.3 source (`occrom13.asm`, `IBMSG`) is `DB 'Osborne System ONE.'` and then `DB ' MONITOR 1.3  '`; and `research/roms/os1-144.rom` at `0x1C3` is `OSBORNE 1`.

If the upper line reads `Osborne System ONE.`, stop and dump it — that is the ROM nobody has.

1a. The second free test, and the one that matters most: press `^D`

The early ROMs carry a **resident diagnostic section**, entered from the boot prompt with `^D` (the 1.3 source says so in `occrom73.asm`'s header). The later ones **do not have it at all**. Checked against every ROM we hold:

ROM	`^D` diagnostic menu?
rev B, 9/12/81	**yes** — `ROM DIAGNOSTICs`
1.3, 12/16/81	**yes** — `ROM DIAGNOSTICs`
1.40	**no**
1.43	**no**
1.44	**no**

So, at the boot prompt, before pressing RETURN:

- **a `ROM DIAGNOSTICs` menu appears** → the ROM is an **early** one: rev B, 1.2, 1.2.1 or 1.3. Combined with the title test above, that excludes rev B and 1.3 — so it is **1.2 or 1.2.1, both of which are `NO_DUMP`.** Dump it.
- **nothing happens** → it is 1.40, 1.43 or 1.44, all of which we already hold. Nothing is gained and the trip is not worth making.

The menu is bright text, so the faulty dim bit does not hide it, and the test needs no disk, no tool and no typing beyond `^D`. The 1.3-era menu reads `A,B Boot Sys / D isk / K ey-vdt / M emory / R ead in test`; a *different* set of entries is itself evidence of an even earlier ROM than 1.3.

2. Reading the bytes: the ROM is *not* visible from CP/M

This is the part that catches people out. `DUMP.COM` and `DDT`'s `D` command will **not** show the ROM: the monitor ROM is banked into `0000–0FFF` only when `ROM MODE*` is selected, and a CP/M program runs with the RAM bank in place. The mechanism is in `docs/o1-memory-io-map.md` §2 and §4, measured on the running machine by `tools/check_o1_map.sh` (#64):

- `OUT (0),A` selects **bank 2** — the ROM at `0000–0FFF`, the memory-mapped I/O window at `2000–2FFF`, and **`4000–FFFF` still bank 1 RAM**;
- `OUT (1),A` selects **bank 1** — main RAM;
- the port number selects and the data byte is ignored;
- the selection is **re-derived**, not latched (`tools/o1_mame_probe.lua` check C10: write the RAM bank, let emulated time pass, and the ROM bank is back) — so the bank is selected *and used* inside the copy, never assumed to hold;

- an interrupt acknowledge reselects the ROM bank, so **`DI` for the duration of the copy**, or a 60 Hz video interrupt will put RAM back under the read;
- the program itself must live **above `4000`**, because with bank 2 selected `2000–3FFF` is the I/O window and code there would vanish.

So reading the ROM is a three-instruction preamble plus a copy: select bank 2, copy `0000–0FFF` to somewhere in `4000+`, select bank 1, return.

3. What is on the disks, so nothing is planned around a missing tool

Checked against the in-repo images, both **single-density**:

Disk	Tools
`OS1SYSS.IMD` — *CP/M 2.2 System Disk*, SSSD 92K	`autost copy help movcpm pip setup stat sysgen xdir` — **no `ddt`, no `asm`, no `dump`**
`OS1UTLS.IMD` — *CP/M 2.2 Utilities Disk*, SSSD 92K	`asm autost ddt dump ed install load movcpm stat submit sysgen xdir xsub`
`OS1XUTLS.IMD` — *Extended utilities* (80-col card)	`autost copy movcpm pip setup sysgen xdir`

So: **`DDT` is on the Utilities disk, not the System disk.** Boot the System disk, then swap the Utilities disk into drive A and run `DDT` from it. (CP/M is already in RAM; nothing is written to either disk.)

4. The DDT procedure (needs only the two SD disks above)

`DDT`'s `A` command assembles straight into memory, so the program is *typed in mnemonics* — no assembler, no source file, no disk write. Type it at `0x4000` and it copies the ROM to `0x5000`, where `DDT`'s `D` can show it.

```
A>DDT
-DDT
-A4000
4000 DI
4001 XRA A
4002 OUT 0
4004 LXI H,0000
4007 LXI D,5000
400A LXI B,1000
400D MOV A,M
400E STAX D
400F INX H
4010 INX D
4011 DCX B
4012 MOV A,B
4013 ORA C
4014 JNZ 400D
4017 MVI A,1
```

```
4019 OUT 1
401B EI
401C RET
401D                     <- a blank line ends assembly
-G4000
-D51C0,5200             <- the banner, copied
```

Bytes, so the assembled listing can be checked by eye:

```
4000 F3        DI
4001 AF        XRA A
4002 D3 00     OUT 0        ; bank 2: ROM at 0000-0FFF
4004 21 00 00  LXI H,0000
4007 11 00 50  LXI D,5000
400A 01 00 10  LXI B,1000
400D 7E        MOV A,M      <- loop
400E 12        STAX D
400F 23        INX H
4010 13        INX D
4011 0B        DCX B
4012 78        MOV A,B
4013 B1        ORA C
4014 C2 0D 40  JNZ 400D
4017 3E 01     MVI A,1
4019 D3 01     OUT 1        ; bank 1: back to RAM
401B FB        EI
401C C9        RET
```

`D51C0` then shows `0000+1C0`′s bytes — the banner. `MONITOR 1.3` in hex is `4D 4F 4E 49 54 4F 52 20 31 2E 33`; `OSBORNE 1` is `4F 53 42 4E 4F 52 4E 45 20 31` — either is unmistakable in a dump.

How far this is checked. The mechanism is measured — `docs/o1-memory-io-map.md` §4 was measured on the running machine with `tools/check_o1_map.sh` — and the byte offsets in §5 come from the shipped ROMs. The DDT route itself is *reasoned* from those and has **not** been run end to end: the attempt under MAME failed on the harness, `tools/research/o1keys.lua` segfaulting partway through typing the program in, not on the program.

5. Fingerprints, so the answer is not a judgement call

Offsets are into the 4096-byte image.

ROM	Offset	Bytes	Text
1.44	`0x1C3`	`4F 53 42 4F 52 4E 45 20 31`	`OSBORNE 1`
1.44	`0x1F7`	`52 65 76 20 31 2E 34 34 …`	`Rev 1.44 (c) 1983 OCC`
1.43	`0x1F7`	`52 65 76 20 31 2E 34 33 …`	`Rev 1.43 (c) 1982 OCC`
1.3	`0x19E`	`4F 73 62 6F 72 6E 65 20 53 79 73 74 65 6D 20 4F 4E 45 2E`	`Osborne System ONE.`
1.3	`0x1D2`	`20 4D 4F 4E 49 54 4F 52 20 31 2E 33 20 20`	`MONITOR 1.3`
1.3	`0x0F4`	`63 2E 20 31 39 38 31 20 4F 43 43`	`c. 1981 OCC`
rev B	—	from `occrom1b.asm`	`OSBORNE SYSTEM ONE.` (caps)

And check the encoding of the *messages*, not just the text. The banner is plain ASCII in every revision. What differs is how the **message strings** are written: rev B and 1.3 use the `DBE`/`DCE` pseudo-ops, so each byte is stored **doubled** (`value * 2`, which is why `ROM DIAGNOSTICs` appears as `40 40 40 A4 9E 9A …`), while 1.44 uses `DB`/`DC` throughout and stores plain ASCII. Shift a doubled string right one bit to read it. See `docs/rom-13-reconstruction.md` §4.

The 1.3 offsets come from the reconstructed image, and **not** from the damaged regions (`0x1F0`, `0x8B0`, `0xA80`, `0xAB0`, `0xFA0`, `0xFD0` — see `docs/rom-13-reconstruction.md` §7), so they are trustworthy. `c. 1981 OCC` at `0x0F4` is a `COPYR:` label the 1.3 ROM never references — dead data, but a unique and very convenient fingerprint.

A rev-B image from `OCCROM.IMD` (`OCCROM0B.COM`, CRC32 `f5f697c7`) is also on record, and it boots — `docs/o1-rom-variants.md`.

6. A dead end, recorded so it is not repeated

`SETUP.COM` does not identify the ROM. It offers *"press M to get source from memory"*, which reads the running system's **CBIOS configuration block**, not the monitor ROM — its six items (printer, baud, screen size, auto-scroll, function keys, arrow keys) are the CBIOS's, and its own banner (`Rev 2.2`) is `SETUP`'s version, not the ROM's. Checked by running it under MAME. The ROM and the system disk version independently; that is the same trap as reading `CBIOSV` out of the CBIOS.

7. For a real dump — what MAME actually wants

Everything above is a *software copy through the machine*: fine for identification, and usable as evidence, but it is not a chip read.

UD11 is a socketed 2732. Pulling it and reading it in an EPROM programmer is the only true dump, and it needs no software, no banking subtleties and no serial transfer. That is the artifact to hand upstream; the DDT copy is the thing to do first because it takes two minutes and tells you whether the trip is worth making.

Getting a 4 KB file off the machine otherwise means either transcribing a hex dump from the screen, or a serial capture — the ROM's own SIO init is `SI.S16`, "16x clock for 1200 baud" (`occrom13.asm`), and CP/M's `PIP` can send a file to `PUN:`/`LST:`, but binary down a 1200-baud line wants a cable and care.

7a. The machine this was written for is out of play

The single-density machine that prompted this doc is **out of play** — it blew a RIFA capacitor in the PSU. The `^D` test in §1a was **never run**, so its ROM was **never identified**, and the question of whether it held a `NO_DUMP` revision is still open. The other two machines on hand are 1.44, which is already held (`docs/o1-rom-variants.md`), so there is at present **no hardware route to any `NO_DUMP` slot** — §1a and §4 are procedures waiting on a machine, not instructions for one.

One repository fact belongs with it. The tree holds the supply's **repair manual** — `research/osborne1/o1astecpsac8151.pdf`, the *Repair Manual for Astec Power Supply AC8151*, with the test set-up, the 115/230 V input-select wire and the +5/+12/−12 rail table (#52; a row in `research/README.md`) — but **no supply schematic and no board-level view**, and the Technical Manual in-repo (`research/osborne1/2F00153-01_Osborne1TechnicalManual_1982.pdf`) carries only **§1.2 "Power On"**, two pages of procedure. What is missing is exactly what a photograph of the board would supply. The one design note is `docs/design/o1next-lunchbox-concept.svg`, which records that the original supply "and its RIFA caps" are retired in that concept.

8. What the result would settle

- **A 1.3 dump fills `osb13.bin`** and closes the `NO_DUMP` slot outright.
- **It would also grade the reconstruction in `docs/rom-13-reconstruction.md`.** That doc's control build reproduces a shipped Osborne ROM to 3759 of its 4096 bytes, and the same toolchain's 1.3 image is uncertified for the same reason; a real dump would say whether the recovered `DBE`/`DCE` rule, the `occtxt6.ast` assumption and the 28-byte question were right — a *positive* answer would validate the method for rev B, BIOS A, 1.2 and 1.2.1 as well, and a negative one would say exactly where the method is wrong.

Related: #205, #70, #72, #222, `docs/o1-rom-variants.md`, `docs/o1-memory-io-map.md`, `docs/rom-13-reconstruction.md`.

OZROM 1E: a third-party monitor ROM

Status: Solved — as tooling, and **#81 is closed** on that deliverable. It boots under MAME on all three Osborne 1 machines (measured, §3), its 58-page manual is OCR'd and indexed here (§4–§8), and its three engineering features — a software 52/80/104-column switch, a chip-level memory tester and a *documented* ROM interface — are available to us without hardware. The column switch's mechanism is measured from the ROM as well as documented (§7): it drives the stock video PIA through the stock ROM's own helper, and OZROM adds no cell of its own. Its

companion **Utilities disk**, which surfaced in 2026-10, is catalogued in §10 — the directory **and all twenty file bodies** are complete. One residue is routed rather than carried: **#167** (drive the column keys from the harness), §9. **#168** — filed for a complete Utilities disk — is closed: the "290 of 400 sectors erased" reading that justified it was our own sector-skew bug (§10.2).

Source: `research/roms/OZROM-1E/OZROM_1E.BIN` (4096 bytes) and `OZROM_1E_Manual.pdf`, both from BrettHallen's `Osborne_1` repository; `OZROM_1E_Manual.ocr.txt` is the 300 dpi `tesseract` text layer (grep it for the appendices — but not for exact code, it is OCR). Micro Management, Inc., Manual Rev. A 84/09/01. Serves #81, and touches #24 (video), #29 (validation), ##45/#47 (keyboard), #20 (floppy) and #7/#18 (clock).

1. What it is

A replacement for the 2732 monitor ROM at UD11, sold in 1984 by Micro Management (Irving, TX). It is a **ROM swap with capability**, not a font change: the manual's feature list is mostly things the stock ROM lacks. The two engineering features that matter to this project are the *documented memory map and jump table* (§5–§6) — a second, independent description of the machine's interface to hold against `rom144.asm` (#72) — and the *software column switch* (§7), which is on the design path for the video personality (#24).

2. How to run it under MAME

MAME cannot be told about a BIOS it does not define, but it **will** load a file whose bytes do not match the expected hash (it warns and carries on). So present OZROM under a stock BIOS filename.

Build a romset, do not just drop files in a directory. The simple `mkdir .../osborne1 && cp ...` recipe in `mame-emulation.md` §5 works for `osborne1` and `osborne1sp` but **fails for `osborne1nv`**, because the Nuevo machine needs `character_generator_6-29-84.14`, which MAME satisfies by CRC fallback from `7a3007-00.ud15` *only when the files are loose in the rompath root or inside a zip* — not when they sit in a plain `osborne1/` directory. Build the zip instead:

```
mkdir -p /tmp/ozbuild/osborne1
cd /tmp/ozbuild/osborne1
cp <repo>/tools/emulator-setup/roms/*.ud11 \
   <repo>/tools/emulator-setup/roms/*.ud15 \
   <repo>/tools/emulator-setup/roms/*.ua15 .
cp <repo>/research/roms/OZROM-1E/OZROM_1E.BIN 3a10082-00rev-e.ud11
cp 7a3007-00.ud15 character_generator_6-29-84.14   # satisfies the clone's
chargen
mkdir -p /tmp/ozset && zip -q -X /tmp/ozset/osborne1.zip *

mame osborne1nv -verifyroms -rompath /tmp/ozset     # -> romset ... is good
```

The `character_generator_6-29-84.14` copy is not a hack: it is a byte-identical duplicate of `7a3007-00.ud15` (`docs/o1-rom-variants.md`), and it is what makes the hash fallback deterministic across all three machines.

3. Measured: it boots on all three machines

```
O1_ROMPATH=/tmp/ozset tools/research/mame-boot-probe.sh MACHINE \
    tools/emulator-setup/floppies/52-blank.imd 120
```

Machine	Screen	Columns	Banner	Result
`osborne1`	416×240	52	`OZROM 1E #00187 / (c) 1984 Micro Management`	**BOOTED** to `A>`
`osborne1sp`	832×240	104	same	**BOOTED** to `A>`
`osborne1nv`	640×240	80	same	**BOOTED** to `A>`

So OZROM's *window* is the machine's (52/104/80), exactly like every other disk (`mame-emulation.md` §2), and it boots CP/M off the utility disk to `A>` on all three. MAME prints *"the machine might not run correctly"* because the ROM hash is not what it expects — that warning is the point, not a problem.

4. What it adds, and what it gives up

From the manual (page 1 feature list; Appendix C for the losses):

Gained	Why it matters here
Keyboard-selectable **52 / 80 / 104-column** display	A *documented software* video-mode switch — the mechanism the video personality (#24) is being built on. On an un-expanded machine it can also toggle the left/right 52 columns of an 80-column display.
System memory tester that names the failing chip	A ready-made second oracle for #29, alongside `tools/check_o1_map.sh`.
Software-redefinable keyboard, plus eight locally-redefinable function keys (up to 63 chars)	Relevant to #45/#47.
Time-of-day clock (`TIME`, `ALARM`, `TIMER`, `QUIKTIME`)	#7/#18 — see §8 and `docs/rt60a-analysis.md` "The third shape".
Documented interfaces: interrupt hook, ROM jump table, memory map, keytable	An externally documented O1 ROM interface to hold against `rom144.asm` (#72).
1793/1797 FDC notes; truer Televideo 912/920 emulation	#20 and terminal behaviour generally.
Removes the *"Centronics printer not connected"* lockout on DD systems	#19.

Given up	Consequence
IEEE-488 functions (Appendix C)	**OZROM and Drive C cannot coexist out of the box** — Drive C is an IEEE-488 device. The manual says *"extra software would be required"*, and that software is the driver whose source we hold in `rom144.asm` (#6/#22).
The stock 4 KB video window	OZROM uses **3 KB** (`F000–FBFF`) for video and the last 1 KB (`FC00–FFFF`) for the redefinable keyboard and function keys, so it scrolls by moving the 23 lines above rather than the window. This is the 1 KB price of the redefinable keyboard, and it constrains the 2.0 ROM design (#83).

The IEEE-488 loss is not passive: the jump-table appendix says the entries at `013F–0156` *were* the IEEE-488 routines and **calling one now produces a controlled system crash** (no disk damage — the machine's two top display lines go "gangbusters hog-wild"), and *"IEEE-488 AS IOBYTE DEVICE 3 NOW NULL DEVICE"*. The parallel/Centronics entries at `0193+` survive unchanged.

5. The OZROM memory map (Appendix C-5)
"Some of the memory locations reserved by the OZROM 1E for its own purposes." The OCR splits the address column from the description column, but the pairing is unambiguous; the clock rows are confirmed independently against the OCC source in `docs/rt60a-analysis.md`.

Address	Purpose
EF07	Parallel-printer control — 0 = normal, non-zero = null device (DD only?)
EF57–EF58	Current BIOS starting address — how to find the start of the O1′s special BIOS regardless of MOVCPM/OS patches
EF5C–EF5D	KTBPTR — keyboard pointer. OZROM sets it to FC00; if (KTBPTR)+2 changes from its initial 0, OZROM repoints it at the ROM table
EF5F	Keyboard matrix value of the last key pressed
EF66	**Start of system clock — hours, 2-digit BCD**
EF67	**Minutes, 2-digit BCD**
EF68	**Seconds, 2-digit BCD**
EF69	**Sixtieths of a second, 2-digit BCD**
EFDF	Time-display toggle byte
EFE0	Keyclick toggle byte
EFE1	Keyclick counter
EFE2	Blink toggle byte
EFE3	Blink counter
EFE4	Blink rate value — gives 30/X blinks/sec; default 20 (decimal) = 1.5 blinks/sec
EFE5	KEYTYP — whether the current key is normal / shift / control / control-shift
EFE6–EFE7	Temporary pointer used by the keyboard-redefinition routines
EFE8	Clock position — F000 + ((value + 43) MOD 256), anywhere on the first two display lines
EFE9	Temporary storage of the character at the cursor (used by the blink routine)
EFF2–EFF4	**Interrupt routine hook** — preset with RET (0C9h); see §8
F000	Beginning of video RAM (3 KB, F000–FBFF)
FC00	Start of the RAM keycode table: FC00–FC37 normal, FC38–FC6F shift, FC70–FCA7 control
FCA8–FCBF	Reserved for future use
FC[illegible]–FD[illegible]	**[illegible]** (64a[illegible])

6. The ROM jump table (Appendix C-6)

Provided "for the programmer's convenience … since actual ROM addresses called by this table may vary according to which version of the OZROM 1E is being used". This is the *same interface* the stock ROM exposes at `0100+`, so it is directly comparable to `rom144.asm`'s vector table (`#72`).

Addr	Entry	Note
0100	JMP COLD_BOOT	
0103	JMP WARM_BOOT	
0106	JMP INPUT_STATUS+01	keyboard
0109	JMP CONSOLE_INPUT+01	keyboard
010C	JMP CONSOLE_OUT+01	display
010F	JMP LIST	serial output
0112	JMP PUNCH	serial output
0115	JMP READER	serial input
0118	JMP DRIVE_RESET	
011B	RET ! NOP ! NOP	**no ROM-level select-disk**
011E	JMP READ	
0121	JMP WRITE	
0124	JMP READ_TRACK_INFO	
0127	JMP READ_SECTOR	
012A	JMP WRITE_SECTOR	
012D	JMP LIST_STATUS	serial-port status
0130	JMP SENSE_DISK_DENSITY	same entry as the stock ROM's SENDEN
0133	JMP ROM_JUMP_1	exit from the interrupt routine for bank 1
0136	RET ! NOP ! NOP	no ROM_JUMP_2
0139	JMP FORMAT	
013C	JMP INIT_SERIAL	
013F–0156	*(were IEEE-488 routines)*	**calling these crashes the machine**
0157–0186	same functions as O1 ROM 1.4	
0187	OR 0FFH ! RET	IEEE printer input status → **null device**
018A	OR 0FFH ! RET	IEEE printer output status → null
018D	XOR A ! RET ! RET	IEEE printer input → null
0190	RET ! RET ! RET	IEEE printer output → null
0193	JMP PARALLEL_OUTPUT_STATUS	Centronics parallel output; needs RWCMD (EFD1) and SAVTYP set

7. The keytable (Appendix C-7) and the column switch

OZROM's normal, shift and control keytables are moved up into RAM at power-up or RESET (`FC00+` per §5); the control-shift table stays in ROM and cannot be redefined. The codes are the machine's own, not ASCII: `0x80–0x89` are the ten function keys, `0x8A–0x8D` the four arrow keys, and the control-shift table carries the "process immediately" codes `0xF0–0xFC` (screen positioning; the time, keyclick, blink and auto-horizontal-scroll toggles; the **Screen-Pac 52/80/104-column select keys**; and the assign/reassign-parallel-printer keys). Three codes are permanently assigned: `0xDD` = ctrl-shift-Z, `0xDE` = ctrl-shift-X, `0xDF` = ctrl-shift-R.

The column switch is the part on our design path (#24). OZROM's own words (Appendix C): the ctrl-shift keys include "the Screen-Pac 52-, 80-, and 104-column select Keys", and on a machine with no 80-column upgrade the switch "can also toggle between the left and right 52 columns of an 80-column display".

Measured against the three machines (§3)

OZROM boots at the machine's native width — **52 / 104 / 80** columns on `osborne1` / `osborne1sp` / `osborne1nv` — and reaches `A>`. That is the same result every other disk gives (`mame-emulation.md` §2): the *window* is the machine's, the *text layout* is the software's.

What is **not yet measured** is the switch itself in action: pressing ctrl-shift- and watching the width change. Doing that needs the harness to post a *modifier-qualified* keystroke (`tools/research/mame-boot-probe.lua` only posts plain characters), and the column keys' effect under MAME depends on whether the driver models the width as a PIA-driven horizontal offset or as fixed hardware. It is the one open item from #81 that needs code rather than reading — routed to **#167**, see §9.

The mechanism, from the ROM

The switch is **not a new mechanism and it uses no new RAM cell**: it drives the video PIA through the *stock* ROM's own helper.

`rom144.asm` defines `H.VIO = 02C00H ; Video memory controls` and `PIAAD = 0EF61H`, and carries this routine:

```
LDK  A,4+3          ; PIA control: select the data register
STO  A,H.VIO+1      ; 2C01 = 7
MOV  A,C
STO  A,PIAAD        ; 0EF61 - the stock ROM's "last PIA-A data" cell
STO  A,H.VIO        ; 2C00 - send data
RET
```

Disassembling `OZROM_1E.BIN` finds **the same routine, unchanged**, at `016F` (`ld a,007h / ld (02c01h),a / ld a,c / ld (0ef61h),a / ld (2c00h),a / ret`), and OZROM's console path reads the same two registers back at `04EC` and `0BE1`. So the column keys compute an offset and hand it to the stock video-PIA write: the **register that holds the mode is `2C00`**, with **`EF61` (`PIAAD`)** as the cached copy — which is why OZROM's own memory map (Appendix C-5, §5) has no column-mode byte in it. There is not one.

Two things this does **not** settle, and they are the honest boundary of the finding:

- **Which field** of the `2C00` byte is the horizontal offset. That port also carries the drive-select and density (`DDEN`) lines (`o1-memory-io-map.md` §3), and `rom144.asm`'s own note on `PIAAD` is about **bit 0 = density**. The manual gives the *effect* precisely — control-up-arrow repositions the window to start at column 29, control-down-arrow to column 76, control-`[` homes it — so the values are known even though the bit layout is not.
- **The dispatch address** for the control-shift codes `0xF0–0xFC`. A linear sweep cannot follow it: `z80dasm` warns about self-modifying code, and the codes are dispatched through a table rather than by direct calls. Locating it is worth doing only if we want to *trace* OZROM's handler; for #24 we need the register and the values, and those are above.

8. The clock, the memory tester, and the error codes

Time-of-day clock. OZROM's `TIME HHMMSS` sets it, ctrl-shift-T shows it (top two display lines, position set by `EFE8`), and `ALARM`/`TIMER`/`QUIKTIME` ride the interrupt hook. Its cells are the machine's own `EF66–EF69` (hours, minutes, seconds, sixtieths, BCD) — **not** a new ABI. Accuracy is "better than three minutes a day" with the drives idle; disk access momentarily stops the clock (the manual is explicit that the clock "has no guarantee" under disk load). The full treatment, and why it matters to our own clock contract, is in `docs/rt60a-analysis.md` "The third shape".

The interrupt hook. On the 60 Hz video interrupt, OZROM's `SYSINT` saves the stack pointer, PC and registers and then calls **`0EFF2h`**; a bare `RET` there continues normally, so a programmer puts a jump to their routine at `EFF2` and ends with `RET` — **no register save/restore needed**. 26 bytes of interrupt stack (13 levels) and 192 bytes of routine RAM (`FCC0–FD7F`) are available. Note this is *not* the RT-60A's `0xEFF8` hook: different `INTBL` slot, different calling convention (see `rt60a-analysis.md`).

The memory tester. Press TAB at power-up or after RESET; it cycles `0 → 0EFFFh`, writing and checking `0`, `0FFh`, `0AAh` and `55h` at each location and naming the failing chip. Video RAM is not tested. A ready-made second oracle for #29.

Error codes (Appendix D): `BOOT ERROR` = "you are not using an OZROM 1E"; `NO SYS!` = a formatted diskette with no system on it (SYSGEN it); `OH, NO!` = same as `BOOT ERROR` (or a bad system track). Also worth knowing: OZROM **enables CRC checking on disk access** where previous ROMs did not, so copy verification now catches real read errors.

9. What is left, and where it goes

#81 is closed on its stated deliverable. The manual is indexed, the column switch is documented *and* its mechanism measured from the ROM (§7), the memory tester and the error codes are written down (§8), and the ROM interface is recorded — memory map, jump table, keytable, interrupt hook (§5–§6). Four residues were left at the close, and the table records where each one went; only the first is still live, and it is routed rather than carried because it needs code rather than reading:

Residue	Where it goes
Drive the column switch from the harness (§7) — the one measurement still missing	**#167** — the harness can only post plain characters, so ctrl-arrow and ctrl-shift are inexpressible. That is a tools capability, wanted by #24 as much as by this issue.
A complete copy of the Utilities disk (§10)	**#168 — closed: we already have it.** The "290 of 400 sectors erased" reading was a `cpmfs.py` sector-skew bug (§10.2); with the correct skew all twenty files extract whole.
Diff the jump table against `rom144.asm` (§6)	**Done** — see below.
Reconcile the `EF66–EF69` clock against ROM 1.44	Not OZROM's question: it is about *which stock revision* dropped `UPTIM`. Recorded in `docs/rt60a-analysis.md`; belongs with #83/#18, not here.

The jump-table diff, done

Entry for entry, the manual's Appendix C-6 table (§6) and `rom144.asm`'s vector table at `0100`+ describe **the same interface**. Every address matches, including the ones where the *names* differ: `0118` is the stock ROM's `JMP RDRV` / `HOME` and OZROM's `DRIVE_RESET`; `0112` is the stock ROM's second `JMP LIST` for punch output, which OZROM calls `PUNCH`. The only structural differences are the two already recorded — **`013F–0156`**, where OZROM removed the IEEE-488 routines so that calling them crashes the machine, and **`0136`**, where the stock ROM's `JMP ROMJP2` becomes `RET / NOP / NOP`. Two independent descriptions of the same interface agreeing is the point of the exercise.

10. The OZROM 1E Utilities disk — the software half (#81)

`research/roms/OZROM-1E/OZROM_1E_Utilities.IMD` is the disk that goes with the ROM: the utilities the manual documents, as a customer received them. We had the ROM and the manual and not the software, so this is the missing third of the package.

It is a **data disk, not a system disk.** It does not boot — the system tracks carry a stub whose message is "There is no CP/M system on this disk." You boot your own CP/M and put this in drive B, which is why the package shipped as a bare set of files.

10.1 The directory — complete

Readable in full: 20 files, all in user 0, in creation order, allocation intact. Standard single-density O1 geometry (40 cyl × 10 × 256 B FM, 2 KiB blocks, 3 boot tracks, 64 directory entries).

File	Bytes	File	Bytes
ALARM.COM	640	OZPATCH.COM	3328
CLKPATCH.COM	640	QUIKTIME.COM	256
CLKPATCH.DOC	1792	QWERTY.COM	1024
CRASH.COM	256	READ.ME	1280
CRCKLIST.CRC	512	SET.COM	2048
DVORAK.COM	1024	SHOW.COM	512
DVORAK.DOC	1024	TD.COM	896
FK.COM	2816	TD.DOC	1024
MEMORIZE.COM	1280	TIME.COM	640
MYKEYII.COM	640	TIMER.COM	768

The set maps onto the manual's feature list one for one: the column switch (`SHOW`), the keyboard remaps (`DVORAK`, `QWERTY`, `MYKEYII`), the clock (`TIME`, `TIMER`, `QUIKTIME`, `CLKPATCH`, `ALARM`), the function keys (`FK`, `MEMORIZE`), and `OZPATCH` to patch the ROM image itself. Two things it settles that the manual only implies:

- **The utilities are separate `.COM` files, not ROM calls.** Nothing here is resident; the ROM supplies the interface and the disk supplies the programs. A reproduction does not need to ship this disk to be OZROM-compatible.
- **`CRCKLIST.CRC` is a self-check file** — a name/CRC list, so a 1984 owner could verify their copy. It is the vendor's own integrity manifest, and the hashes are what a replacement would be checked against (§10.2 has the news: we do not need one).

10.2 Condition — intact: the file bodies are whole

The disk is whole. Its ~291 sectors of uniform `0xE5` are the **format fill** a CP/M formatter writes across a disk — **free blocks, not holes** — and the arithmetic agrees to within a track: twenty files totalling ~22 KB, plus the directory and the system tracks, is ~110 sectors of the 400, which is what is not `0xE5`. Re-extracted with the correct geometry, **all twenty files are whole**: no file holds an interior run of `0xE5` longer than a handful of bytes (those are ordinary data bytes), and every size matches the directory. `READ.ME` is complete from its first byte, which the emulated machine confirms by printing it from the start (§10.2a).

Measured how. The file bodies come out from sectors that were never written if the read uses the wrong sector order. The `osb1sssd` preset in `tools/cpmfs.py` needs the format's **sector skew** (`skew 2` — the Osborne SD layout interleaves its ten sectors per track); without it, block pointers resolve to the wrong physical sectors: the *directory* still reads, because the three directory sectors in use happen to land inside block 0 in some order, while every **file body** comes back as `0xE5`. `cpmtools`' own `osb1sssd` diskdef carries the skew, and reading the same image through it returns real content — `CRCKLIST.CRC`, for one, is intact text. `osb1sssd` carries `skew=2` (**#170**), and `cpmfs.py`

generates the logical→physical table the same way `cpmtools` does (`cpmfs.c`): for this format, `0 2 4 6 8 1 3 5 7 9`. The directory read goes through the same mapping, so there is no off-by-one phantom entry. Checked against `cpmcp -T raw -f osb1sssd` on all three in-archive SD images — this disk, `DRIVE_C.IMD` and `OCC1_HARDDISK.IMD` — where **every file matches byte for byte**. Nothing else in the archive rests on the bad read: the two other `osb1sssd` extractions (`research/drive_c/extracted/`, `research/harddisk/extracted/`) were made with `cpmtools`, which always had the skew, and are unchanged, and non-interleaved formats default to `skew 1`, so `kp2x` and `drivec` output is unchanged. **This disk was the only casualty.**

10.2a Measured on the machine: the ROM makes no difference

The OZROM ROM and this disk are independent of each other, and running them together changes nothing about what the disk holds. The same image under the stock ROM and under OZROM (`-flop1 52-blank.imd -flop2 OZROM_1E_Utilities.IMD`), with the same commands posted to each:

	stock ROM	OZROM 1E
`DIR B:`	20 files	**the same 20 files**
`TYPE B:CRCKLIST.CRC`	the full vendor CRC list	**the same list**
`TYPE B:READ.ME`	the TIMER syntax note, from its start	the same

So the diskette does **not** read differently with the ROM installed. The ROM supplies the interface and the disk supplies data; nothing in OZROM's disk path changes what the bytes are. OZROM *does* enable CRC checking on disk access (§8), but that can only make it stricter, not more permissive.

One behavioural difference is worth knowing, and it is not about reading: **OZROM's console acts on its control-shift codes `0xF0–0xFC`** — the "process immediately" set from §7 — so `TYPE`ing a *binary* file such as `CRCKLIST.CRC` makes OZROM execute whatever codes its bytes happen to contain. Under OZROM the screen came up with the keyboard-redefinition display after that `TYPE`; the stock ROM simply prints the bytes.

10.3 The boot sector is not vendor code

Track 0 sector 1 holds a stub, and it does not look like anything Micro Management shipped:

```
C3 5C D2   JP 0D25Ch
...
+20   "PUT THIS ON T0S1"
...
+80   "THIS IS A TEST OF THE AMERICAN BROADCAST NETWORK"
```

"PUT THIS ON T0S1" is an instruction addressed to whoever was writing the disk, and the Emergency Broadcast System test line is a fill/verification pattern. The pair reads like a **placeholder written by whoever produced this image**, not product. Two consequences: treat the boot sector as an imaging artefact rather than evidence about the product, and expect that this disk cannot be booted to check anything — that has to be done by copying the files onto a system disk.

10.4 A tooling bug this found

`imdinfo.py --raw` appended sectors in the order they appear in the `.imd`, which equals logical order only when the numbering map is sorted. Single-density Osborne disks are not: they number `(2,3,…,10,1)`, so **every track came out rotated by one sector.**

The failure was silent, and that is the part worth remembering. A rotated CP/M disk still parses — the directory lands somewhere plausible and `cpmfs.py` returns a file list — it is just the *wrong* list. Here it returned 12 files instead of 20, with 8 of them carrying another file's blocks. Nothing errors, nothing looks malformed, and the output is plausible enough to archive.

Fixed in `tools/research/imdinfo.py`; the dump is now byte-identical for sorted maps (checked against `DRIVE_C.IMD`) and correct for rotated ones. Two archived images turn out to be rotated and were never convertible before the fix: `research/diagnostics/DIAG_2.04.IMD` now yields the **15-file Osborne diagnostic disk** (`DIAG.COM`, `FILETEST.COM`, `DDBIT.COM`, `SETUP.COM`, `SYSGEN.COM`, …), and `research/source-code/OCCROM.IMD` its 77-track FM/MFM disk.

A second bug of the same class does more damage. `cpmfs.py`'s `osb1sssd` preset had no **sector skew** for the format it names, so every SD Osborne *file body* it read came back from the wrong sectors — the read that made this disk look "partially erased". §10.2 has the mechanism, the fix (**#170**) and the check against `cpmtools`. The lesson is the same for both bugs, and it is the reason neither was caught by a test: both produced output that *parsed*, and the result looked like physical damage rather than a misread. **A reader that returns plausible garbage is more dangerous than one that errors.**

10.5 What it changes for #81

Less than the file count suggests: the ROM is the deliverable and this is the optional companion disk. What it does give us:

- The **complete and authoritative file list** of the utilities — and, as §10.2 records, the **complete files themselves**, not a holed copy.
- Confirmation that the utilities are **external programs, not ROM entry points**, which bounds the reproduction scope: OZROM compatibility means the ROM's jump table and memory map, not this disk.
- A second, independent description of the feature set to hold against the manual (§4).

The hunt for a complete copy is closed. #168 was filed to find one and closed the same day — the "290 of 400 sectors erased" that justified it was our own skew bug (§10.2), and the disk we hold is whole. The one check still not possible is against the vendor's own manifest: `CRCKLIST.CRC` lists a two-byte CRC per file, and none of the standard CRC-16 variants tried so far reproduces them, so per-file verification against the manifest has to wait for someone to identify the algorithm.

Sources

- `research/roms/OZROM-1E/OZROM_1E.BIN`, `OZROM_1E_Manual.pdf`, `OZROM_1E_Manual.ocr.txt` — BrettHallen `Osborne_1`, `ROM/`; OCR per the `research/README.md` convention (300 dpi, `tesseract`).

- `research/roms/OZROM-1E/OZROM_1E_Utilities.IMD` — the companion utilities disk (§10). Owner-supplied local file, 2026-10-11; source URL not yet recorded. ImageDisk 1.19, label `ozrom`, 40 cyl × 10 × 256 B FM.
- `research/source-code/BIOS-144-source/rom144.asm` and the OCC CBIOS modules (`occram25.asm`, `occbio15/45/95.asm`) — the stock interface and RAM map the jump table and clock cells are compared against.
- `docs/mame-emulation.md` §2/§5, `docs/o1-rom-variants.md`, `docs/rt60a-analysis.md` — machine widths, ROM inventory, the clock.

ROM 2.0: CP/M resident in the ROM

Status: Partially solved. The ROM half is settled and measured — **7695 bytes** for an interposer + ACB-4000 boot with no floppy read, ever (§2.3); the **53-entry jump table at `0100h`** and the `ROMVEC+3N` contract that pins its numbering (§3); the payload split and the boot path, built and booted in `osborne1nx` (§4.3, §4.6); the **150-byte monitor budget** (§4.7); and the **feature set decided against OZROM's** — what to take and what to refuse (§4.8). The **hardware half is open**: there is no ROM 2.00 part (§4.1), and the window's address, paging and selector are **#12′s interface spec** (§4.1a). §8 lists what is left, item by item.

The answer. ROM 2.0 is an O1 ROM that boots the machine to a usable prompt **without reading a floppy** — real or virtual. It replaces the monitor's *boot path*, not its *interface*: the stock ROM's 53-vector jump table at `0100h` and the CBIOS's `ROMVEC+3N` arithmetic stay exactly as Osborne shipped them (§3), and the boot keys the build needs are re-lettered in place for two bytes (§4.6). The boot itself is the **ACB-4000 hard-disk path**: `RETURN` runs the vendor's own loader block out of the ROM window, and the MD-10 BIOS reads CP/M off the drive, so the machine reaches `A0>` with **no floppy in it** — measured in the emulated machine (§4.6, §7). What it costs is **7695 bytes**, the one number for this boot (§2.3), served from the interposer's **softROM** rather than from a part in the ROM socket (§4.1).

What is left is the hardware half and every check that needs a real machine: the window's address, paging and selector belong to #12 (§4.1a), and each acceptance below is a MAME run — nothing here has been run on a physical O1 (§8).

Issues: #83 (deliverable 1; deliverable 2′s toolchain proof landed as #203/#204), ##49 (the reproduction this ROM is for), #72 (the ROM source), #179 (the MD-10 boot that wants the same ROM window), #12 (the window's address, paging and selector), ##247 (the window, the door and the boot screen as built), #278 (the ROM-side frame path, retired), #24 (the column mode), #81 (OZROM, the pattern), #22 and #19 (the peripherals the feature set refuses to give up), #29 (the memory test as a second oracle).

1. What ROM 2.0 is, and what it is not

The goal from #83 is an O1 ROM that boots to a working `A>` with **no disk access at all**. What this project builds is the half of that goal that survived: a boot with **no floppy read, ever** — real or virtual — which the ACB-4000 hard-disk path reaches (§2.3). CP/M resident in the ROM with no disk of any kind is the larger goal, and it was dropped as a product decision (§4.4). ROM 2.0

is a *replacement part*, not a patch: optional, socket-reversible, legitimate on the replica and (with the interposer's ROM daughterboard, below) on a real O1.

One ROM, two hosts. The interposer carries a lead-set to the **main ROM socket (UD11)** as well as the character-generator socket (UA15), so the ROM window is the interposer's to fill on any OEM-faithful mainboard, and the SFF board grafts the same capability in. **There is one ROM 2.00** — one image, one layout — and it runs on either.

That is also why there is **one host brief**: the interposer's ROM window does not care which OEM-faithful mainboard it is plugged into — an existing OEM board, or any faithful 1:1 recreation of one with the Dual Density upgrade built in. (Such a recreation is *possible* if a chassis ever needs restoring; it is not in the set, and it has no directory — see *Hardware that is not a board* in `hardware/README.md`.) So "the stock machine" in this document means the mainboard-plus-interposer combination, and there is no second path to design for.

OZROM 1E is the pattern, not the feature set. #83′s direction, and the measured shortcomings in `docs/ozrom-1e.md`:

Borrow — structural	Why	Explicitly not wanted
A *documented* ROM jump table and memory map	OZROM's manual is why third-party code can target it; the stock ROM already has a table (§3), it is just undocumented	OZROM's keyboard redefinition, which takes `FC00–FFFF` of video space. #24′s VRAM shadow assumes `F000–FFFF` and OZROM's move constrains every later decision
A software 52/80/104-column switch	The mechanism already exists in the stock CBIOS (§4.7, `SCRMOD`) and it is on #24′s design path	—
A chip-level memory tester	The O1 has no built-in RAM test; it is cheap and it is what a new ROM is for	—

2. The arithmetic, measured

The four pieces, all read from primary sources or the shipped bytes:

Piece	Size	Where it lives	How we know
Monitor ROM	**4096**	`0000–0FFF`, bank 2	The part: 2732 at UD11. Its own source asserts its code ends below `0FFBh`
CCP	**2048**	`CB00–D2FF`	`occbio15.asm` pins `CCP = 0CB00h`, `BDOS = ccp+806h` ⟶ the CCP is `0x800`
BDOS	**3584**	`D300–E0FF`	`BIOS = ccp+1600h` and the CCP/BDOS boundary at `ccp+806h` ⟶ `1600h − 800h = 0E00h`
CBIOS	**2048**	`E100–E8FF`	`BIOS = ccp+1600h`, `MSIZE = 59` ⟶ the BIOS region is `E100–EBFF`; its own guard stops code at `CCP+1E00h = E900h`
Total resident	**11776**		**11.5 KB**

This is the *no-disk-anywhere* list — CP/M resident with no disk of any kind, real or virtual. It is **not** the list for the boot this project is building, which is shorter because the ACB-4000 supplies the CCP and BDOS: see §2.3 for that list and its single total.

The ROM's own boot code states the total, and it agrees. `rom144.asm`'s `CBOOT` and `WBOOT` load the system in 128-byte blocks, and the counts are the arithmetic:

Routine	Constant	Bytes	Equals
`CBOOT` (cold boot)	`LDK B,60 ;CCP/BDOS/CBIOS`	7680	2048 + 3584 + 2048
`WBOOT` (warm boot)	`LDK B,44 ;CCP/BDOS and don't read CBIOS`	5632	2048 + 3584

That is the strongest evidence available for the 11776-byte figure: the shipped firmware says it reads 7680 bytes of "CCP/BDOS/CBIOS", and 4096 + 7680 = 11776.

2.1 The numbers are measured on a shipped disk too

`research/shipped-software/OS1SYSD.IMD` (DD, 1024-byte sectors, 5/track, 3 reserved tracks) carries the system image from the **CCP** upward, and every piece lands exactly where the source's equates say:

Image offset	Address	Content	Size
`0000`	`CB00`	CCP	2048
`0800`	`D300`	BDOS — entry at `D306` holds `JP 0D311h`	3584
`1600`	`E100`	CBIOS — its 34-entry jump table, byte for byte	2048
`1E00`	`E900`	`CCP+1E00h`: the CBIOS's own *RAM* area begins here	—
`2000`	`EA00`	end of what SYSGEN wrote (8 sectors, rounded up)	—

Four independent anchors, none of them assumed:

1. **The CBIOS's jump table** is at image offset `1600`, and `occbio15.asm` puts it at `BIOS = CCP+1600h`. Its 13 `CALL ROMCDE` entries are 13 consecutive `CD 12 E3` at `E139h–E15Ah` — one routine, called thirteen times, exactly as the source's table is written.
2. **That routine's address is confirmed by module order.** `occbio35.asm` places `ROMRI`, then `ROMCDE`, `ROMCD1` and `ROMJMP`; the image has `ROMRI` at `E2FEh` (20 bytes of `POP`/`EI`/`RET`), `ROMCDE` at `E312h`, `ROMCD1` at `E317h` (`16 01` = `LD D,01h` = `high(ROMVEC)`) and `ROMJMP` at `E319h`.
3. **The ROM locates the CCP itself, at every boot, from the same bytes.** `CBOOT` reads sector one to `0D000h` and checks three bytes of it: `[0D000h] == 0C3h`, `[0D001h] == 05Ch`, and then `[0D002h] - 3` is the CCP's page. On the disk those bytes are `C3 5C CE` — and `0CEh - 3 = 0CBh`, so the CCP is at `CB00h`. The first three bytes of the CCP are therefore a *marker* as well as code (`JP 0CE5Ch`), which is why `0x5C` is checked.
4. **The `ROMCDE` arithmetic requires it.** It forces the high byte of the vector to `high(ROMVEC) = 01h` and leaves the low byte as the CBIOS's own address low byte, so `BIOS & 0FFh` must be `00h` — `E100h` does, `E000h` would not.

A caution, because it nearly fooled this pass. The 512 bytes at image offset `1E00–1FFF` are *not* CBIOS code. They are the contents of RAM at `E900h` when the disk was made — DDT's leftovers, written out because SYSGEN writes whole sectors. On this disk they happen to be a readable Osborne sales text ("… Accessories planned for the Osborne 1: 12" display monitor … battery pack for portable operation …"). Anyone who reads the system image as"the CBIOS" will count 2560

bytes and be wrong by 512. The 2048 figure is the source's own guard: `IF XXX > CCP + 1E00H / .9 ERROR SYSTEM SIZE TOO LARGE`.

2.2 Why 4 KB cannot hold it, and why the ROM page is not one block

Bank 2 is the only place the CPU reads ROM (`docs/o1-memory-io-map.md` §2):

Range	Bank 2
`0000–0FFF`	Monitor ROM (2732 at UD11)
`1000–1FFF`	Unused — **mirror of the ROM**, because A12 is not decoded
`2000–2FFF`	Memory-mapped I/O
`3000–3FFF`	Unused — mirror of `2000–2FFF`
`4000–FFFF`	Mirrors bank 1 RAM

So the ROM page offers **8 KB contiguous** (`0000–1FFF`) and **4 KB more at `3000–3FFF`** — but only by jumping *past* the assigned I/O block at `2000–2FFF`, so the two runs are not one block.

The boot this project is building fits the contiguous 8 KB. Interposer + ACB-4000, to `A>`, with no floppy ever read, is **7695 bytes** (§2.3) — 497 bytes to spare, and no paging. The 11776-byte figure in §2 is a *different* list for a *different* goal (CP/M resident with no disk of any kind): it is the one that fits neither run, it is the one that needed §4.1a's paging, and the goal itself was dropped (§4.4). There is no 16 KB ROM in any case, and never will be (§4.1). The stock 2732 holds 4096 of which **150 bytes are free** (`0F6Ah–0FFFh`; the code ends at `0F69h` and the source's own guard is `IF RLWA > 0FFBH`).

2.3 The contents, and the one number

This is the goal the project is actually building: an **interposer + ACB-4000** that boots to `A>` on the hard disk with **no floppy read, ever** — virtual or physical. It has one content list and one total.

Piece	Bytes	Why it is there	Source
Monitor ROM	**4096**	the machine's ROM. Its boot path is replaced; its 53-vector jump table is kept	`rom144.asm`
CBIOS	**2048**	**the MD-10 BIOS borrows 30 of its 34 jump-table entries.** Measured: `HARD.COM` copies `18 + 21 + 3 + 3 + 45 = 90` bytes from `(page zero+1)+3 = 0xE106` — CBIOS vectors 2–31 — into the MD-10 BIOS's slots at `0xD806`, `0xD86A`, `0xD82D`, `0xD87F` and `0xD833`. Those are `JMP`s into the CBIOS, so the CBIOS's code has to be in RAM for them to land anywhere	`OS1SYSD.IMD`, image offset `0x1600`
MD-10 BIOS image	**1477**	the ACB-4000 driver, and the partial BIOS that reads the system off the hard disk. Copied to `0xD800`	`HARD.COM`, file offset `0x053A` (`0x5C5` bytes)
install / patch / jump	**60**	the vendor's own loader block — *the shim*, in §4.6′s language — kept in the window as the reference `HCBOOT` was lifted from: `LD HL,053Ah` / `LD DE,0D800h` / `LD BC,(0547h)` / `LDIR`, the five patching `LDIR`s, `XOR A` / `LD (0013h),A`, `JP 0D800h`. **Nothing calls it** — the machine forces `HCBOOT` to do that work in the monitor in	`HARD.COM`, `0x01E5`–`0x0220`
the hard-loader stub	**715**	the one piece only RAM can run: `LD HL,0CB00h` / `LD (0EFD2h),HL` / `LD A,80h` / `LD (0003h),A` /	`tools/research/rom2image.py`
Total	**4345**		

7695 bytes — and it fits the 8 KB contiguous window with 497 bytes to spare. No paging is needed for this boot at all.

What is deliberately *not* in the list. The **CCP and BDOS**, 5632 bytes, because the ACB-4000 supplies them: `HARD.COM`'s BIOS reads 11 blocks from **LBA 5** into `0xC200` and jumps to it (`docs/md10-system.md` §3–§5). Carrying them in ROM as well would be a second copy of bytes already on the drive, and would make the ROM decide which one wins.

Nor are the loader's other 1022 bytes. `HARD.COM` is 2560 bytes — a 1082-byte loader and a 1478-byte embedded BIOS, of which 1477 are copied. The loader's banner, its presence probe and its `TEST UNIT READY` are diagnostics for a human installing a card; the MD-10 BIOS's own `BOOT` selects the card and reads. The shim does not need them.

One number, and it is this one. `11.2 KB` and `11776 bytes` are two *different lists*, and neither is the boot described here: 11.2 KB is monitor + MD-10 + CCP/BDOS with **no CBIOS**; 11776 is monitor + CCP + BDOS + CBIOS with **no MD-10**. The boot described here is **7695**, and any figure quoted for it has to name the list it came from. **The boot screen does not come out of the 497-byte spare, and does not need to.** `research/boot-screen/` holds the authored art and `tools/research/rom2strokes.py` measures it: every raster-shaped encoding of the logo lands at **784–857 B** — raw striped bitmap 857, RLE 816, an exact rectangle cover 784 — because the letters are heavy and the cover is dominated by diagonals (the X is 72 rectangles where it is structurally two strokes). The geometric route was not converged, so it is estimated rather than demonstrated.

What settles it is the architecture: **one window page is 4096 bytes and the video buffer is 128 × 32 = 4096 cells**, so **one page is exactly one screen**. In bank 2 the window and the video RAM are both visible (§4.4), so a frame is one `LDIR` from `1000h` to `F000h` — about 40 bytes of ROM, with the frames in the interposer's flash, and the animation is the viewport slide, which is a register write. The one casualty is that the art is 153 columns and a screen is 128, so the 26 columns of hatch to the left of the O are dropped in favour of keeping the logo whole.

3. The interface that already exists

ROM 2.0 does not have to invent its jump table. The stock ROM **already publishes one**, at `0x0100`, and the CBIOS already calls it — Osborne simply never documented it outside the source.

Measured: 53 entries, `0100h–019Eh`, 3 bytes each. The table ends exactly where the source's `IMSG:` sign-on message begins (`019Fh`), which is what makes 53 the whole table rather than a guess — the first byte of `IMSG` is `'Z'-40h = 1Ah` and `019Fh` holds `1Ah`. `tools/research/romvec.py` prints the table, labels each entry from `rom144.asm`'s own table text, and cross-checks the binary's opcode sequence against the source's — **0 disagreements** on the 1.44 ROM.

#	Addr	Target	ROM source	CBIOS vector	Note
0	0100	0255	CBOOT	CBOOT	cold boot — **this is the one ROM 2.0 replaces**
1	0103	02AB	WBOOT	WBOOT	warm boot — **hard-coded by `occbio55.asm`**
2	0106	0375	SKEY	CONST	keyboard status
3	0109	0382	CI	CONIN	keyboard input
4	010C	03EF	COUT	CONOUT	console output
5,6	010F,0112	0BEC	LIST	LIST, PUNCH	both point at one routine
7	0115	0BD2	READER	READER	reader input
8	0118	0C15	RDRV	HOME	home the drive
9	011B	—	RET/NOP/NOP	SELDSK	**stub** — the CBIOS supplies SELDSK itself
10–12	011E–0124	0D49,0D50,0DBE	READ, WRITE, RADR	SETTRK, SETSEC, SETDMA	**labels do not line up** — see §3.2
13	0127	0C2F	RSEC	READ	one sector read — **hard-coded by `occbio65.asm`**
14,15	012A,012D	0C36,0BE3	WSEC, SLST	WRITE, LISTST	one sector write; list status
16	0130	0C7C	SENDEN	SECTRN	**labels do not line up** — see §3.2
17,18	0133,0136	0099,009E	ROMJP1, ROMJP2	RRI, ROMJMP	RAM/ROM exit; the ROM call
19–23	0139–0147	0600,0D40,0D5…	IEEE…, VEDIR, STODIM	ACRTC	IEEE-488 status/input/output quartet

3.1 How the CBIOS reaches it — and why the numbering is a contract

Two mechanisms, both in the source, both already load-bearing:

1. **By arithmetic, from the CBIOS's own address.** `occbio35.asm`'s `ROMCDE` pops the return address, subtracts 3, and forces the high byte to `high(ROMVEC) = 01h`. So CBIOS vector *N*, sitting at `BIOS + 3N`, selects ROM vector *N*. `occbio15.asm` states the requirement in a comment: *"NOTE: ROM jump vector must match CBIOS vector"*. This is why `BIOS & 0FFh` must stay `00h`, and why the CBIOS's 13 `CALL ROMCDE` entries all land on the right routines without a table pointer.
2. **By absolute address, twice.** `occbio55.asm`'s warm boot does `LDK DE,ROMVEC+3*1`, and `occbio65.asm`'s sector transfer does `LDK HL,ROMVEC+3*13`. The CBIOS also calls the ROM's `SENDEN` by its absolute ROM address (`LDK DE,SENDEN`), which is why **the monitor must stay mapped while the CBIOS is running** — see §4.2.

The consequence for ROM 2.0: vectors 0–52 keep their numbers and their semantics, and the monitor stays where it is. Both of the hard-coded vectors are exactly the two the diskless build wants to change (§4.5), and both are changed *inside the ROM*, so the CBIOS does not have to move.

3.2 Two things that do not line up, recorded rather than smoothed over

- **Indices 10, 11, 12 and 16 carry the ROM's own labels** (READ, WRITE, RADR, SENDEN) where the CBIOS's table has SETTRK, SETSEC, SETDMA, SECTRN. Nothing is broken: the CBIOS implements those four itself and never calls the ROM for them, so the ROM's slots are free to be its own API.
- **`occtxt6.ast` publishes constants for the alignment that does not exist** — `BSTTRK = 10*3`, `BSTSEC = 11*3`, `BSTDMA = 12*3`, `BSECTR = 16*3`, alongside the ones that are correct (`BCONST = 2*3`, `BREAD = 13*3`, `BIECO = 21*3` … `BIEPP = 28*3`). **No CBIOS source uses any `B*` constant, or `BIOCAL`, the macro that would.** They are dead code, and four of them have the wrong numbers. Recorded here because the obvious reading of that file is that they are the mapping.

3.3 What this gives the design

- A **documented interface already exists** — 53 entry points with a stable numbering — so ROM 2.0 extends it rather than inventing one, which is exactly the OZROM property worth borrowing (#81).
- The **warm-boot hook already exists**: the CBIOS's `WBOOT` calls ROM vector 1. A ROM-resident CP/M needs a different *source* for the system image, not a different *mechanism*.
- The **boot keys already exist**, and ROM 2.0 re-letters them (#247 step 4): `RETURN` = **the hard disk**, `:` = floppy A, `"` = floppy B, and `ESC` is released. 1.44 had `RETURN` = floppy A, `"` = floppy B, `ESC` = hard disk — so the machine's own disk sat on a *shifted* key while a facerolled RETURN booted a floppy nobody asked for. `HCBOOT` is a documented empty stub (`JMP START`) that Osborne left in the ROM for exactly this purpose (`rom144.asm` line 4634; it assembles to `JP 0000h` at `0F67h`), and the re-lettering is **two bytes** of the dispatch, in place — §4.6.

4. The design

4.1 The softROM serves the window — there is no ROM 2.00 part

There is no physical ROM 2.00 and there never will be — not a 27128, not a 2764, not a 16 KB part in any form. The ROM window is served by the interposer's **softROM**: the interposer already has the bus, and the ROM socket bridge gives it the window itself, so it presents the ROM in firmware.

The host	**OEM board + interposer**, or the **`next-integrated board`**, where the interposer is built in as one unit. There is no third case and no part in the socket
A mainboard on its own	Keeps a socket that takes a standard 1.44 ROM, so a stock machine boots stock. That is the *stock* part — not ROM 2.00
Window	the monitor at `0000–0FFF`; the payload at `1000–1FFF` — two contiguous 4 KB pages, measured (§4.1b)
Banks	page 1 *is* the payload, and `PRESENT(0)` is what serves it (`docs/rom-pico-protocol.md` §4). The 4 × 4 KB split — system image 2, MD-10 BIOS 1, diagnostics + spare 1 — was the dropped no-disk goal's plan (§4.4)

What this document does not decide. The window's address, how the payload is paged, and what the selector is are **pseudo-hardware**, and they are **#12′s interface spec** — not this document's call. The route here needs none of the three: the Pico already sees every bus cycle and presents the window itself, so no part is fitted and no `A12`/`A13` latch is needed (§4.1a). `2300h` was proposed as that latch's address, and it is not free (§8 item 3).

4.1a The socket's limit, and why it does not bind

The limit is real, and worth stating because it is what *would* bind if we were fitting a part: **a 24-pin 2732 socket has `A0–A11`, `/CE`, `/OE` and the data bus — no `A12`, no `A13`, and no write strobe.** A part in that socket could never exceed 4 KB however large it was.

It does not bind, because **we are not fitting a part.** The softROM is not in the socket's address space the way a part is: the Pico already sees every bus cycle and presents the ROM window itself, so the lines past `A11` are irrelevant — which is also why ROM 2.00 needs **no programmer** and is updatable in flash. With the part removed nothing else drives those data lines either, so the per-cycle bus race #83 raised does not arise.

The socket bridge is not speculative as a technique. `docs/o1-rom-variants.md` records **BrettHallen's 2732→2764 daughterboard image** (8 KiB, CRC32 `edc31846`): OCC 1.44 and OZROM 1E in one 2764, **with `A12` selecting between them** — banking in the ROM socket, built

and tested, passively. That is prior art for the mechanical form, and it is why the bridge is a known-good shape. It is **not** the route ROM 2.00 takes.

Why the larger-part route is not the design. A 24-pin 2732 socket has no way to reach a bigger part, and three things follow: there is no board with no interposer — the `next-integrated` board has one built in, and an OEM board takes one; with no part in the socket there is nothing for `A12`/`A13` to address; and `2300h` was proposed as a *latch* address, so with no latch it would be a candidate register address — and it is not free (§8 item 3).

What is open, and where it lands. The window's address, the payload's paging and the selector are **#12's interface spec** (§4.1) — not decisions this document makes. What is settled here is the constraint from §2.2: the ROM page gives 8 KB contiguous plus 4 KB beyond the assigned I/O block, 11776 bytes must be presented for the no-disk goal, and there is no larger part to put them in.

4.1b The window this boot uses, measured in the machine (#247)

§4.1 leaves the window's address, paging and selector to **#12**, and this does not close that. But the boot this project is actually building (§2.3, **7695 bytes**) needs **none of it** — and that is measured in the emulated machine rather than argued.

The window is `0000–1FFF`, and it is already 8 KB wide. The mainboard never decoded `A12` (§2.2), so the stock machine reads the same 4 KB monitor twice across that range. MAME says so in one line:

```
m_rom_view[0](0x0000, 0x0fff).mirror(0x1000).rom().region("maincpu", 0).unmapw();
```

`mirror(0x1000)` *is* the undecoded `A12`. So a softROM that decodes it gets two 4 KB pages **contiguous, in the ROM page, with no new address space, no latch and no register** — which is exactly the "8 KB contiguous" of §2.2, and why the 7695-byte image fits with 497 bytes spare.

Page 0 (`A12 = 0`)	the monitor, `0000–0FFF` — the 4 KB that must never move (§4.2)
Page 1 (`A12 = 1`)	the payload, `1000–1FFF` — 3599 bytes of the 4096 (§4.3)
Given a 4 KB image	`A12` is ignored and the page mirrors: the stock part, unchanged

What that makes step 1 of #247. The machine already has the idiom for replacing a view entry: `osborne1sp` calls `osborne1_mem(map)` and then re-registers the entry it wants to own — and the same shape replaces the ROM window:

```
m_rom_view[0](0x0000, 0x1fff).r(m_softrom, FUNC(softrom_device::read)).unmapw();
```

So **for this boot there is no window decision to wait on** — only the file the device serves, and whether an image is 4 KB (mirror, stock) or more (two pages). A latch at `2300h` would be for a *part*; the softROM needs none, and this is where the window stops being an argument and becomes an address map.

And it is built and measured. `softrom_device` is `src/devices/machine/softrom.{h,cpp}` in the patched tree (`tools/emulator-setup/mame-osborne1nx/`), `osborne1nx_mem()` is the one line above, and the image is named on the machine's own command line after the device's tag — `-softrom rom2.rom`. What the window serves:

Image	The window
none	the machine's own ROM region, mirrored — a stock machine
4 KB or less	the image, mirrored: the image *is* the stock part
larger	two pages, `0000-0FFF` and `1000-1FFF`; past the image's end, `0xFF` (erased flash)

`tools/emulator-setup/mame-osborne1nx/softrom-check.sh` is the acceptance, and it is all of it:

- **the window is the device's** — `o1next-softrom.lua` reads all 8192 bytes back and compares them against the table above. This half is not optional: the stock monitor never reads page 1, so an override that reached only `0000-0FFF` would boot identically and silently.
- **the boot is identical** — `osborne1nx` with no image mounted, and with `rom2image.py`'s 7695-byte image mounted, both produce a screen dump byte-identical to `osborne1`'s, `A>` included (the machine name is the only difference).

Measured 2026-10-14, and again on 2026-10-15 with the hand-over in the image: all five checks pass. The 7695-byte image reads `31 C1 EF D3` at `0000h` (the monitor) and `C3 ED E4 C3` at `1000h` (the CBIOS, which is `JP`s), with the hand-over stub's `21 00 CB 22 D2 EF 3E 80 32 03 00 C3 00 D8` at `1E01h` and `FF` from `1E0Fh` to `1FFFh` — the payload's 497-byte spare. The MD-10 hard-disk boot to `A0>`/`B0>`/`C0>` is unchanged.

4.2 The invariant that keeps the CBIOS working

The monitor is never banked. Only the window moves, and for this boot it is `0000–1FFF` (§4.1b). This is not a convenience — it is required, because the CBIOS calls ROM routines by *absolute address* (`LDK DE,SENDEN` in `occbio45.asm`, `LDK DE,ROMVEC+3*13` in `occbio65.asm`) while bank 2 is mapped, and the CBIOS's own `ROMCDE` arithmetic (`ROMVEC+3N`) depends on the monitor being where it always was. Bank the monitor and the CBIOS breaks in ways that look like data corruption.

The other invariant follows from it: **`0000–0FFF` must stay a 4 KB monitor**, so ROM 2.0′s new monitor code has to fit in the **150 free bytes** of the 1.44 layout. That budget is in §4.7, and it is the real design constraint of this whole document.

4.3 The content, and what it costs

The **payload** is what goes in the window; the monitor is not part of it. For the boot this project is building (§2.3) the payload is **3599 bytes**:

Holds	Size	Why
CBIOS	2048	the running BIOS whose vectors 2–31 the MD-10 BIOS patches in (§4.6). Resident in RAM at `0xE100`
MD-10 BIOS image	1477	copied to `0xD800` — the ACB-4000 driver, and the partial BIOS that reads the system off the drive
install / patch / jump	60	the shim — the vendor's loader block at `HARD.COM 0x01E5–0x0220` (§4.6). The machine forces `HCBOOT` to do that work in the monitor, so nothing calls this copy; it is carried as the reference
the hand-over stub	14	the store only RAM can make: `CCPADR`, the IOBYTE, and the MD-10 BIOS's entry. `HCBOOT` copies it to `0xE0F0` and the bank switch enters it (§4.6)
spare	497	`8192` (the contiguous window) – `4096` (monitor) – `3599`
payload total	**3599**	**7695 with the monitor** (§2.3)

The no-disk goal's payload was a different, larger one — the system image, 7680 bytes (CCP + BDOS + CBIOS) — and *that* is the one that does not fit the contiguous window (§2.2). It is not in the build, because the goal is dropped (§4.4); it is recorded here because it is the one that needed a paging answer.

Where any of it is presented is measured — the payload is window page 1 (§4.1b) — and what the rows above are is content slots, not addresses.

4.4 Cold boot — the dropped no-disk path

The no-disk goal is not a goal any more, and the decision is a product one rather than a technical one. What the project wanted was *"it turns on and is sitting at a useful prompt"* — **no floppy in the drive to kick off the boot-transfer** — and that is reached without this section: the firmware serves drive letters (the interposer's own storage personalities), so there is always a disk, and the MD-10′s `HCBOOT` path (§4.6) boots the hard disk with no floppy in the machine at all.

Three consequences. The **11776-byte payload and its paging are out of the plan**; SYSLD and §4.5′s vector-1 retarget have nothing left to do; and the largest open item in §8 item 1 is #12′s general window rather than this boot. The boot this project is actually building is HCBOOT, the ACB-4000 path (§4.6), and that is the one §2.3 sizes. What follows is the cold boot for the dropped goal, kept as the record of why it went.

The copy semantics below are the dropped goal's design; "latch bank *N*" was a placeholder for a selector that is now window page 1 (§4.1b). The destinations, the LDIR lengths and the final jump do not depend on how the window is paged.

The copy is cheap because of a property of bank 2 that is easy to miss: **in bank 2, 4000–FFFF mirrors bank 1 RAM.** So while the monitor is running from the ROM window it can write the RAM at CB00h+ directly — source and destination are both visible at once, and no RAM/ROM mode switching is needed mid-copy.

1. START prints the prompt (unchanged).
2. RETURN/" → the ROM's CBOOT path is replaced by **SYSLD**: latch bank 0, LDIR 4096 bytes from 1000h to CB00h; latch bank 1, LDIR 3584 bytes from 1000h to DB00h. Roughly 30 bytes of code.
3. JP 0E100h — the CBIOS's CBOOT, **still in bank 2.** No bank switch here, and that is deliberate: in bank 2 0E100h is already bank 1 RAM, so the jump lands on the freshly-copied CBIOS, and the CBIOS calls SW2RAM itself (its first CALLs) exactly as it does today. Switching to RAM *before* the jump would be a bug — it would unmap the monitor out from under the very instruction that follows.
4. The CBIOS's CBOOT does what it always did: MAGIC (screen size, via SW2ROM/02400h), the IOBYTE, the baud rate, CDISK = A, page zero (0000: JP WBOOT, 0005: JP BDOS), then JMP [CCP+3] with C = drive.
5. The CCP prompts **A>** — and nothing has touched the FDC.

Note what does *not* change: HCBOOT becomes SYSLD with a different bank selection, RETURN still boots the machine's own system — the one key that keeps its meaning across the two goals, §4.6 — and the CBIOS is byte-for-byte the vendor's. The copy destinations also land exactly where the source's own RAM equates put them: 0CB00h + 7680 = 0E900h, which is CCP+1E00h, the first byte of the CBIOS's RAM area.

4.5 Warm boot

Warm boot needs no change, because ROM vector 1 keeps its meaning. The CBIOS's WBOOT calls **ROM vector 1** (LDK DE,ROMVEC+3*1), and with the firmware serving drive letters there is always a system on A:, so it reads CCP+BDOS from the disk exactly as the stock ROM does. The CBIOS does not change at all, and the mechanism is the one CP/M already uses on this machine.

The ROM-resident alternative — vector 1 pointing at SYSLD, or at a SYSLD variant that copies only the CCP+BDOS, 5632 bytes, matching the stock WBOOT's B,44 — belongs to the dropped no-disk goal (§4.4). It is where #83′s "warm boot has no system tracks" question was answered: the ROM's own jump table was always the hook.

One runtime consequence: the CBIOS's disk vectors are still there and still real. `HOME` is harmless (it only flushes a buffer and clears flags), but a program that *selects* a drive will reach `SELDSK` → `SENDEN` → the ROM's FDC spin loop and retry. The clean fix is a **null-DPH gate** (§5, question 4) rather than deleting the drivers.

4.6 HCBOOT, filled in

`HCBOOT` is `JP 0000h` at `0F67h` — three bytes, and the ROM's own end-of-code guard sits right behind it. It becomes, essentially byte for byte, what `HARD.COM`'s loader does once its diagnostics are dropped — **60 bytes**, measured at `HARD.COM` `0x01E5–0x0220`:

```
HCBOOT:  the MD-10 payload is window page 1, always    ; §4.1b
         LD   HL,053Ah                    ; the embedded BIOS in the ROM image
         LD   DE,0D800h                   ; where the MD-10 BIOS lives
         LD   BC,(0547h)                  ; 0x5C5 = 1477 bytes, read from the
image
         LDIR
         LD   HL,(0001h)                  ; page zero+1 = the running BIOS's
WBOOT
         INC  HL / INC HL / INC HL        ; +3 -> vector 2
         LD   DE,0D806h / LD C,12h / LDIR  ; 18 bytes -> 0xD806
         LD   DE,0D86Ah / LD C,15h / LDIR  ; 21 bytes -> 0xD86A
         LD   DE,0D82Dh / LD C,03h / LDIR  ;  3 bytes -> 0xD82D
         LD   DE,0D87Fh / LD C,03h / LDIR  ;  3 bytes -> 0xD87F
         LD   DE,0D833h / LD C,2Dh / LDIR  ; 45 bytes -> 0xD833
         XOR  A
         LD   (0013h),A                   ; page zero, drive A
         JP   0D800h                      ; the MD-10 BIOS's own jump table
```

Two things in that are load-bearing, and both are read out of the file rather than inferred:

- **The patch source is `(page zero+1)+3` — the *running* BIOS's jump table, not the ROM's.** `LD HL,(0001h)` reads the `WBOOT` vector, so on a stock boot the source is the **CBIOS at `0xE100`**, and the 90 bytes copied are its vectors 2–31. That is why the CBIOS has to be resident (§2.3), and why it cannot be deleted from the build as "not needed on a hard-disk machine".
- **The jump is to `0D800h`, the BIOS's jump table**, not to `0D882h`'s `BOOT` directly — entry 0 is a `JP` that gets there.

The copy, the patch and the jump are measured; presenting the payload needs no selector write at all, because it *is* window page 1 (§4.1b).

Built, and measured in the machine. ROM 2.0′s monitor is `rom144.asm` plus one diff — `research/source-code/BIOS-200-source/rom2.diff` — assembled by the vendor's own ACT80 (`tools/research/rom2asm.sh`, the pipeline #203 proved). The result is checked against the invariant the design rests on: **the stock 1.44 below `0F67h` is byte-identical** (3,943 bytes), and `HCBOOT` itself is **77 bytes**, `0F67–0FB3` — 67 bytes of copy, patch and jump, plus the 10-byte `LDIR` that puts the hand-over stub in RAM. With the boot screen's own 26 bytes behind it the whole new tail is `0F67h–0FCDh`, **103 bytes of 149, 46 free** to the vendor's guard (§4.7).

Three things the machine forced, and none of them is visible in the source:

- **Page zero is the ROM in bank 2, so the shim's `LD HL,(0001H)` cannot be used as it stands.** `HCBOOT` runs from the monitor, in bank 2, where `0000–1FFF` is the ROM window: a write to page zero is dropped and a read returns *this file's* bytes — `(0001H)` reads `C1 EF`, the operand of the monitor's own `LD SP,0EFC1H`, not a BIOS vector. `HARD.COM` had to ask page zero where the BIOS was; the ROM knows, and uses the CBIOS's vector 2 at `0E106H`, which is what `(page zero+1)+3` resolves to on a booted system (§2.3).
- **The bank switch has to be taken by code that survives it.** The monitor's own code is in the bank it is leaving, so `HCBOOT` ends by pushing the MD-10 BIOS's entry and jumping to the CBIOS's `SW2RAM` (`0E8E2H` in the shipped bytes): `SW2RAM` switches the bank, and its `RET` lands on the pushed address.
- **The machine's own interrupt exit needs a store only RAM can make.** The ROM's 60 Hz handler returns *through the CBIOS* at `CCPADR + 1633h`, and `CCPADR` is the ROM's own `CBOOT`'s to write — `HCBOOT` was lifted from `HARD.COM`'s loader without it, so the exit lands on `1633h` and the machine never comes back (measured below). It cannot be written from bank 2 either, because `0EFD2h` is reached only in RAM mode — so `HCBOOT` copies a **14-byte stub** out of the payload's spare into RAM at `0E0F0h`, pushes *that* address instead of `0D800h`, and `SW2RAM`'s `RET` enters it. The stub writes `CCPADR`, the CBIOS's IOBYTE default, and jumps to `0D800h`.

`tools/emulator-setup/mame-osborne1nx/rom2-check.sh` measures the chain, and it passes: the CBIOS is at `0E100H` and is the window's `1000H`; the MD-10 BIOS's jump table is at `0D800H`; it is patched from the CBIOS's vectors, all 90 bytes into the five slots; the bank switches to RAM; the stub is in RAM at `0E0F0H` with `CCPADR = CB00H`; and with a drive mounted the system is read off it into `0C200H` (`C3 C8 C4 …`) and the machine reaches **`A0>`**.

The hand-over: measured, and it was one cell. The chain completed and the machine still did not reach `A0>`, and the reason was *not* the CBIOS's uninitialised RAM in general. It was **`CCPADR` (`0EFD2h`)** — the cell the ROM's own `CBOOT` writes at `02A3h` and `HCBOOT` was lifted without.

The ROM's 60 Hz interrupt handler ends by returning *through the CBIOS*:

```
;Exit interrupt code via exiting to RAM and then enable or disable ROM code
;depending on the value contained in ROMRAM cell.
    LD  A,ROMRAM    ;IS ROM IN OR OUT?
    OR  A
    JNZ ROMJP1      ;if return to RAM
```

`ROMJP1` is `CCPADR + 1633h` — the CBIOS's `RRI` at `0E133h`, whose `OUT (C),A` (C = `ROMRAM`) **restores the bank**. With `CCPADR = 0` that exit jumps to `1633h`, which in bank 2 is the softROM's page 1: the payload's own copy of the CBIOS's console dispatch. That copy reads `IOBYTE` at page zero+3 — and in ROM mode page zero *is* the ROM, so it reads the ROM's own byte `D3h` (the opcode of the `OUT (03h),A` at `0004h`), whose low two bits are **3: the IEEE device**. The dispatch trampolines through `ROMJMP` into the ROM's IEEE input routine, which waits for a handshake no device answers, for ever.

Measured with `rom2bank.lua` — the same instrument on the same path, before the store was built and after:

	stock boot	ROM 2.0, before the store	**ROM 2.0, with it**
reads of `0003h` (the IOBYTE)	858, **all `80h`** — RAM, CRTOUT	**1, returning `D3h`** — the ROM, IEEE	**11, all `80h`** — RAM, CRTOUT
bank selects	454 `OUT 0`, 492 `OUT 1`	1 `OUT 1` (HCBOOT), 1 `OUT 0` (the trampoline)	6 `OUT 0`, **44 `OUT 1`**
`CCPADR` (`0EFD2h`), and the exit it gives	**`CB00h` → `E133h`**, the CBIOS's `RRI`	**`0000h` → `1633h`**, the payload	**`CB00h` → `E133h`**

`ROMRAM` (`0EF08h`) reads `00h` at the end of *both* runs, because `SW2ROM` clears it on every ROM call and the sample lands inside one; it is 1 at the tick, which is what makes the handler take the `ROMJP1` path at all.

The stock machine survives because its CBIOS brackets every ROM call with `SW2ROM`/`SW2RAM`, so it is in RAM mode whenever the dispatch runs — 492 asks for RAM in a single boot. The hand-over made exactly one, then left the guest to run across a tick, and the tick's handler could not get back to RAM because `CCPADR` was zero. **The bank is not something a guest can hold**; it is re-derived on every instruction fetch and on interrupt acknowledge (`docs/o1-memory-io-map.md` §4), so what the hand-over has to do is make the machine's own exit work — and the third column is that working: the guest asks for RAM 44 times in the 8 s after RETURN, and the IOBYTE answers `80h`, which is the stock machine's own answer.

The fix is one store, and it is proved twice. First by poking it in from outside: with `CCPADR = CB00h` — and the CBIOS's own IOBYTE, `80h`, at page zero+3 — written into RAM just after the hand-over (`rom2handover.lua`), the machine boots to `A0>` **with no floppy in the machine**:

```
                OSBORNE 1
           Rev 1.44 (c) 1983 OCC
   Insert disk in Drive A and press RETURN.
A0>
```

And then by building it: `HCBOOT` copies the stub out of the payload's spare into RAM and switches the bank to *it*, `rom2-check.sh` with a drive mounted reads the stub's 14 bytes back at `0E0F0h` and `CCPADR = CB00h` on the way past, and the prompt arrives from the ROM alone. Two details are worth keeping:

- **`0CB00h` is not this boot's CCP.** This boot puts the system at `0C200h`; the value is the *base the exit adds 1633h to*, and `0CB00h + 1633h = 0E133h` is the CBIOS's `RRI` — the stock CCP-to-CBIOS offset. This boot keeps the CBIOS at `0E100h` exactly as a floppy boot does, so the stock value is the right one.
- **The stub's home is `0E0F0h`** — 14 bytes in the 104-byte hole between the driver's third allocation vector (`0DFA7h` + 241 = `0E098h`, `docs/md10-system.md` §9.1) and the CBIOS at `0E100h`. It has to be at or above `4000h`: in bank 2 anything below that is the ROM, the window or I/O rather than RAM, and a write there is dropped.

The boot keys, and they cost two bytes (#247 step 4). `HCBOOT` hangs off the sign-on dispatch, which is three comparisons, and ROM 2.0 re-letters it *in place* — no code added, no address moved:

Key	Boots	1.44
`RETURN`	the **hard disk** (`HCBOOT`)	floppy A
`"`	floppy **B** (`DSKSWP = 1`)	floppy B — unchanged
`:`	floppy **A** (`DSKSWP = 0`)	unused
`ESC`	nothing — released	the hard disk

`CMP ESC` becomes `CMP CR` and `CMP CR` becomes `CMP ':'` (`rom144.asm` ~295-309; the diff's first hunk, and the two bytes are `0048h` and `0050h` in the image). It fits in place, and it is correct rather than merely lucky: `START` clears `0EF00h-0EFEFh` on every pass — `DSKSWP` (`0EFC7h`) included — so `:` always finds it zero and means drive A, and `"` is the only path that increments it. Nothing else in the dispatch moves: the three `JZ`s and the `JR` are the ones 1.44 has.

ESC gets no new job. It is a non-standard keystroke in MAME as well as a poor choice for a boot key, and it now falls through to the prompt's `JR START` like any unclaimed key. The CBIOS's own `ESCH` cell is a different thing and is untouched.

Because those two bytes are the **only** thing below `0F67h` that moves, `rom2asm.sh` checks them *by value* rather than merely tolerating them: a build whose dispatch is not this map stops with the map in the message, and `rom2boot.lua` reads the same two bytes back out of the *running* monitor.

4.7 The ROM 2.0 jump table, and the 150-byte budget

The legacy table is preserved exactly — 53 entries, same numbers, same semantics (§3.1). ROM 2.0 adds a second, **self-describing** block after it, so third-party code can detect the ROM and find the new calls without guessing:

Address	Content
`0100–019E`	the 53 legacy vectors, unchanged
`019F`	signature, 8 bytes ASCII: `01ROM2.0` (where `IMSG` sits today)
`01A7`	extension count (1 byte)
`01A8+`	the extension table: 3-byte `JMP`s, `ROMVEC2 + 3N`

Proposed extensions — short on purpose, each one justified by an issue:

#	Entry	Does	Serves
0	`SYSLD`	reload the system image from ROM (also ROM vector 1′s target)	§4.5 — the dropped no-disk goal's loader
1	`BANKSEL`	`A` = the payload slot, write the selector	§4.3 — and page 1 *is* the payload, so what the extension selects is the payload and nothing else (§4.1b, `docs/rom-pico-protocol.md` §4)
2	`SCRMOD`	`A` = `0`/`1`/`3` ⟶ 52/104/80 columns, by the stock video-PIA write to `02C00h` with `EF61h` (`PIAAD`) as the cached copy	#24. The CBIOS's own `PLUGH` mechanism (`occbio55.asm`'s `MAGIC`), and the register and routine confirmed against `OZROM_1E.BIN` (§4.8) — **no video RAM taken**, per #83′s constraint
3	`MEMTEST`	chip-level memory test	§1
4	`HDBOOT`	install and enter the MD-10 BIOS from bank 2	#179
5	`VERSION`	`HL` = version word	so a program can branch on the ROM, which is the OZROM property worth having

The budget. The new monitor code has to fit in the 150 free bytes of the 1.44 layout, and the accounting for everything *planned* — the extensions included — is:

Piece	Estimate
SYSLD copy loop + bank latch + entry	~30 B
HCBOOT body (reuses SYSLD's loop)	~30 B
signature + count + 6 extension vectors	~26 B
SCRMOD, BANKSEL, VERSION	~40 B
Total	**~126 B of 150**

Feasible but tight, and the ACT source's own `IF RLWA > 0FFBH / .9 ERROR CODE TOO LARGE` guard will fail the assembly if it is wrong — which is the right place to find out. **Plan B if it does not fit:** the loader moves into bank 3 and is paged in by a ~12-byte trampoline in the monitor, which buys a full 4 KB for loader code at the cost of one bank. MEMTEST is the first thing to move there.

Two of those rows have moved since the estimate was made: SYSLD is the dropped no-disk goal's loader (§4.4), and the boot screen — 26 bytes, and not in the estimate at all — is now in the tail. What the measured fit below has to hold is the signature, the count and the six extension vectors.

Measured, not estimated. The build's own numbers, as it stands and as it got there. Every row is measured with `tools/research/rom2asm.sh`, and the stock 1.44 below 0F67h is byte-identical in all of them (§4.6):

Build	The monitor's new code	Free to the guard at 0FFBh
HCBOOT alone	**77 B**, 0F67–0FB3 — 67 of copy, patch and jump, plus the 10-byte LDIR that puts the hand-over stub in RAM	72
with the frame path — PRESENT(1)/(2)/(3), the trailer checks and the two blits	**212 B** (182 when first measured; #247′s W3 grew the fallback) against the 72 above HCBOOT ⟶ CODE TOO LARGE	—
the screen's code moved out of the monitor into PRESENT(3)	**128 B**, 0F67–0FE6 (130 before the acceptance found two defects in it) — the cold boot, the sign-on hook, and the 51-byte SCRN stub at 0FB4h–0FE6h	21
as it stands, after #278	**103 B**, 0F67–0FCD — the cold boot, the sign-on hook, and the screen's own **26 bytes**: bright fill, fallback print, return	**46**

So §8 item 2 has two answers, and the second is the useful one. *No* — the monitor code did not fit in 150 bytes, and the screen was the piece that proved it. The assembler's own guard said so in its own words (2026-10-15, #247 turn 3):

```
I 1021  .9   ERROR   CODE TOO LARGE..      Program LWA = 1102h.
```

That is `IF RLWA > 0FFBH / .9 ERROR CODE TOO LARGE` doing its job. The answer was to give the screen's code a home rather than squeeze it: a **third content id in the softROM** (`PRESENT(3)`, a 4096-byte blob in the Pico's flash beside the frame, reached by a ~21-byte stub that copies page 1 to RAM and jumps into it — no payload change, and the screen's code is firmware like the frame), the **payload's spare** (`1E0Fh–1FFFh`, 497 bytes, where the hand-over stub lives), or this section's **Plan B** (a banked loader). The blob landed at `4000h–40B6h`, **183 bytes**, `Program LWA = 40B7h`, and one assembly produced both artifacts — `rom2blob.py` turned the `4000h` records into the firmware asset `firmware/rom-service/screen.bin`, and the tool, the asset and the `-bootsplash` slot all retired with the frame path (#278). The per-piece arithmetic is in `research/source-code/BIOS-200-source/rom2-screen.asm`.

And *yes*, once #278 retired the frame path. With the trailer checks and the two blits gone the screen is **26 bytes**, which fits the 72 the stub occupied, so the code came back into the monitor's own tail and the blob, its asset and its tool went with it. The *message* went into `IMSG`'s own slot at `019Fh` — 72 bytes of text where the vendor's 182-byte sign-on message sat, at the vendor's own length, with the vendor's `FF` fill behind it — so nothing below `0F67h` moved, and the **110 bytes** left over there are what the signature and the extension table in this section will take.

Three defects the build found, and all three are read off the ACT80′s own listing (which `L=B:` writes to a file rather than scrolling past on the console). The door helper's `JNZ` tested flags left by its *length* probe rather than by the result code, so every caller took its "no such content" path and the machine printed the stock 1.44 sign-on however well the door answered; the frame's trailer *sum* was read from the half's `0FFEh` instead of the window's `1FFEh` — the ROM's own byte — so the check could never pass. Neither was visible in a boot; both are why `bootscreen-check.sh` measures the character plane byte for byte and the attribute plane bit 7, and why with no splash mounted it measures the fallback instead. And the **85 errors** were not syntax errors: five comment lines that had lost their leading `;` assembled as four illegal mnemonics — three bytes of junk each at `0FB4h` and `0FD9h` — and one unterminated string, while the blob sat *inside* the vendor's `ECHO 0FFFH-RLWA ! DB 0FFH … ENDM` block. `ECHO` **repeats** its body, so the blob was assembled and listed 11 times: ten extra passes over its eight labels, and 80 duplicate-label errors. 85 = 80 + 5. With both fixed the build is `no ERRORs, 487 Labels, Program LWA = 40B7h`.

4.8 The feature set: what to take from OZROM, and what to refuse

OZROM 1E is the pattern for *how* to be a replacement ROM (`docs/ozrom-1e.md` §4–§8). It is also the cautionary tale: **its feature set is shaped by its part.** It had a 4 KB 2732, and the two things it gave up are things this project needs.

Refuse, explicitly — both are 4 KB decisions, not ideas:

OZROM's choice	Why we refuse it
IEEE-488 removed — jump table `013F–0156` *were* the IEEE-488 routines, and calling one now "crashes the machine"; `IEEE-488 AS IOBYTE DEVICE 3 NOW NULL DEVICE`	**Drive C is an IEEE-488 device** (#22, `docs/drive-c-protocol.md`) and we hold the driver's own source (`rom144.asm`, #6). OZROM could not have both. We can
Video window cut to 3 KB (`F000–FBFF`), the last 1 KB (`FC00–FFFF`) spent on the redefinable keyboard	#24's VRAM shadow assumes the full `F000–FFFF`, and OZROM's move "constrains every later decision" (`docs/ozrom-1e.md` §4)

Take — and there is room to take them:

OZROM feature	Take?	Notes
52 / 80 / 104-column switch	**Yes**	On #24′s design path, and it is **not a new mechanism**: the stock video-PIA write, register `2C00` with `EF61` (`PIAAD`) as the cached copy — the *same routine*, unchanged from `rom144.asm`, disassembled out of `OZROM_1E.BIN` at `016F` (§4.7)
Chip-level memory tester that names the failing chip	**Yes**	The O1 has none, and it is #29′s second oracle. TAB at power-up in OZROM; `MEMTEST` is already in §4.7′s extension table
A *documented* jump table, memory map and version	**Yes**	The stock ROM already publishes the 53-vector table (§3); OZROM's manual is precisely why third-party code could target it. The `O1ROM2.0` signature and `VERSION` (§4.7) are the same property
The interrupt hook at `0EFF2h` — a bare `RET` continues, no register save/restore	**Yes**	#18. It is a **different `INTBL` slot and calling convention** from the RT-60A's `0EFF8h` (`docs/rt60a-analysis.md`); reconcile the two, do not merge them blindly
Time-of-day clock in `EF66`–`EF69` (BCD)	**Yes**	#7/#18. The cells are the machine's own, so this is not a new ABI
CRC checking on disk access	**Yes**	Free correctness; the stock ROM does not do it
Removing the *"Centronics printer not connected"* lockout on DD systems	**Yes**	#19, and cheap
Software-redefinable keyboard plus eight function keys	**Not yet**	#45/#47. This is the one feature that costs **RAM** (1 KB at `FC00`–`FFFF`), not ROM, and it is the reason OZROM moved the video window. Do it with a RAM budget, never by taking video space
1793/1797 FDC notes; truer Televideo 912/920 emulation	**Not yet**	#20; not on this box's path

The rule this leaves us with, and its one constraint. Every entry in the "refuse" table is a consequence of a 4 KB part. Ours is served from the interposer's own flash, and the boot needs 7695 of the 8192 contiguous bytes (§2.3) — so there is room in the *image* that OZROM never had.

The constraint is narrower than that, and worth stating precisely: **the 4 KB monitor is still 4 KB**, with 150 free bytes (§4.7), and anything that must be resident there answers to that budget. The clock, the interrupt hook and the CRC check are bigger than that between them, so they cannot be monitor-resident. They do not have to be — the extension table at `01A8h` (§4.7) is the hook for code that lives in the window, and that is what the window is for. **The monitor stays a monitor; the new features live in the window and are reached through the table.**

So no — we do not have to drop a peripheral the way OZROM did. IEEE-488 stays, the 4 KB video window stays, and the test of the design is that nothing in the "take" table above requires giving either one up.

4.9 The boot screen: one page, and a fallback that pays for itself

The screen is **one window page** — 4096 bytes, which is exactly the video buffer (128 × 32 cells). In bank 2 the window and the video RAM are both visible (§4.4), so one `LDIR` from `1000h` to `F000h` is a whole frame: **4096 × 21 T-states = 86016, or 21.5 ms at 4 MHz.** A fiftieth of a second, against a floppy boot measured in seconds. Serving that frame was the ROM's job while the ROM served the frame; since ##278 the frames live in the **Pico's own flash** and the **RP2354B** draws them, so the ROM's whole contribution to the screen is the bright fill below, and the animation is the viewport slide — a register write.

The frame is someone else's file, so it validates itself. The flash is updatable and a hand-built image will get it wrong, and a bad frame puts garbage on the screen with no way back. So the frame carries a marker in the rows nobody can see — rows 24–31 are off-screen:

offset	content
`0FF8h`	`O1NEXT`, the magic — **six** bytes
`0FFEh`	the 16-bit sum of `0000h`–`0FF7h`, little-endian

The trailer is eight bytes: the magic is **six** bytes, which consumes the two reserved bytes at `0FFEh`, so the checksum definition does not move — it still covers exactly the 4088 bytes before the trailer. The one cost is that the monitor's compare is six bytes rather than four, inside the 150-byte budget §4.7 accounts (#228).

The fallback is the screen's own two lines, and it costs nothing — because ROM 2.0 *replaces* the stock sign-on rather than adding to it. It carries the **same text and the same bright/dim runs as the animation's resting frame**, at the frame's own rows and columns — array rows 20 and 22, ink from columns 24 and 30 (#247′s W3):

	bytes
the message: `VCLRS`, the padding as `OSTR`'s `7Fh` repeat escape, both lines, and the `ESC,VSHI`/`ESC,VEHI` mode changes	72
`IMSG`, which it replaces (§2.1)	182

So the fallback still **frees 110 bytes**, and it needs no code of its own: the attribute runs *are* the text's own escapes, and dim is the vendor's own half-intensity mode. Three things were measured rather than assumed, and the first is why an unpositioned fallback is not merely misplaced — it is invisible:

- **`CURS` is 0 when the screen code runs.** `START` clears `0EF00h–0EFEFh` on every pass, and `COUT` writes through `CURS` — so a message that does not position itself is written to address 0, which is the ROM, and nothing appears anywhere. The vendor's `IMSG` began with `VCLRS` for exactly that reason; the fallback does too.
- **dim is `ESC,VSHI` … `ESC,VEHI`**, the vendor's half-intensity mode: `VHALF` writes `DIMBIT` (`00h`) into the attribute array, which reads back **`7Fh`** — the animation's own dim value, so the plane and the asset's attribute half compare byte for byte.
- **the `7Fh` repeat escape is `OSTR`'s own** (`IMSG` uses it), so a padding run of twenty costs three bytes rather than twenty.

The order at boot is unchanged: **print the text first, then page the frame in and check the trailer; if it is right, blit over the text.** A missing or malformed image leaves the message on screen, which is the whole point of having it.

And the blit is two passes, not one — measured 2026-10-15. The O1′s video cell is **9 bits**: 8 for the character and 1 for brightness, and the 9th bit is not in the video RAM. It lives in a *separate* 4K×1 array that `OUT (02h)` pages into `F000–FFFF`; `OUT (03h)` pages the character RAM back, and the array's low seven bits are not physically present and read high. Measured in `osborne1nx`:

with `OUT (02h)` then writing `00h` to `F000h`	reads back **`7Fh`**
the palette (`palette_init_monochrome_highlight`)	0 = black, 1 = `C0C0C0`, 2 = white
the renderer (`BIT(attr,7) ? 2 : 1`)	**bit 7 set = bright, clear = dim**
the array at the sign-on	4072 × `FFh`, **24 × `7Fh`** — so it is not a blank slate
the 116 character cells at the sign-on	all bright

So the screen **fills the attribute array bright** (`OUT (02h)`, 4096 stores of `FFh`, `OUT (03h)`), because 24 stray dim cells would show in a 4096-cell screen. That is 21.5 ms, and it is what makes the fallback's own dim runs the *only* dim cells on the screen — which is the trigger the RP2354B watches for (§4.9′s `IMSG`, and `docs/rom-pico-protocol.md` §5.1).

And that fill is the only thing the ROM does to the attribute plane. The RP2354B draws both planes out of the frame it is playing (#278), so nothing blits the frame's attribute half over the fill. The frame is still two 4096-byte halves — characters and attributes — and v1.0′s attribute half is all `FFh`. The format, the trailer and the sequence the ROM used to run are `docs/rom-pico-protocol.md` §4.1 and §5.

And the frame lives in the Pico's own flash, not the library flash — one animation, compiled into the firmware, swappable only by rebuilding it. The library flash holds the *default disks*.

`tools/research/rom2strokes.py --frame` writes the whole artifact — **8192 bytes, the character half then the attribute half**, each with its own trailer at its own `0FF8h` — and the art is placed by `--origin`; see §2.3 for why that is not cosmetic. `--half` writes one half, `--verify` validates a half or a whole frame off the disk. What the firmware compiles in is `firmware/rom-service/boot-animation.bin` — the animation the RP2354B plays (#247′s W5) — and `firmware/rom-service/frame.bin` (the single frame) is not in the tree: the animation replaced it. Nor is `firmware/rom-service/screen.bin`, the screen's code served as a window page, because the code came back into the monitor's own tail (#278); `rom2blob.py` and `rom2asm.sh`'s `-b` went with it.

5. The six open questions from #83, answered

#	Question	Answer
1	Where does the BDOS get its parameters with no disk?	**Keep the DPB.** It costs nothing: the DPB, allocation vector and check vector live in the CBIOS's own RAM area at `E900h+` exactly as today, and with no media they are never dereferenced (§4.5, the null-DPH gate). Stubbing them would save no ROM space — they are RAM, not ROM — and would break the case where a real floppy *is* fitted
2	Warm boot, when there are no system tracks	**Moot — there is always a system on `A:`.** The firmware serves drive letters, so warm boot reads the system off the disk exactly as the stock ROM does and **ROM vector 1 keeps its meaning**; no CBIOS change, no new mechanism (§4.5). The ROM-resident alternative — vector 1 pointing at `SYSLD` — belongs to the dropped no-disk goal (§4.4)
3	Does the CCP need to be relocatable?	**No — pin it.** `MSIZE = 59`, `CCP = CB00h`, `BDOS = D300h`, `BIOS = E100h`, all as the vendor shipped them. `MOVCPM`/`SYSGEN` are disk-build tools for a disk build; a ROM build wants the layout fixed so the ROM's copy loop can be two `LDIR`s with literal addresses. It also keeps the CBIOS's `ROMCDE` arithmetic valid, which *requires* `BIOS & 0FFh == 00h`
4	Keep floppy support, or drop it?	**Keep it.** The ROM carries the vendor's CBIOS unchanged, so a real disk still boots. The diskless case is handled at `SELDSK` by returning a **null DPH** when no media is present, which is documented CP/M behaviour (the BDOS then reports `Bdos Err` on a RAW attribute on its own,
5	Which socket/part from the OZROM's memory map?	**The 8kB socket, and it's read-machine** to part of it, a 16kB version, to keep a socket, or manually set its specific choice, provides a RAM attribute on it, over,

The `LDIR`/patch/jump body is **exactly 60 bytes**, not an estimate — measured at `HARD.COM` `0x01E5`–`0x0220` (§4.6) — and presenting the payload costs nothing on top of it, because the payload *is* window page 1 (§4.1b).

6. The build (deliverable 2)

Step	Tool	State
Monitor source assembles to the shipped 4096 bytes	**ACT80** (SORCIM 3.0/3.2, in-repo) under MAME	**Done** — #203/#204, byte-identical, CRC32 `c0596b14`. `tools/research/rom144-reassemble.sh`
The jump table is what the source says it is	`tools/research/romvec.py`	**Done** — 53 entries, 0 opcode disagreements
ROM 2.0′s new monitor code assembles	ACT80	**Done** — the stock 1.44 below `0F67h` is byte-identical, `HCBOOT` is **77 bytes** (`0F67`–`0FB3`), and with the screen's 26 bytes the new tail is `0F67`–`0FCD`, **103 of 149, 46 free** to the vendor's guard. `tools/research/rom2asm.sh` (§4.6, §4.7). CRC32 `ce51a4eb` for the `HCBOOT`-only build
The system image's CCP+BDOS+CBIOS	**taken from the shipped disk**, not rebuilt	Available today — the reserved tracks of `research/shipped-software/OS1SYSD.IMD`, at image offsets `0000`, `0800` and `1600` (§2.1)
The **ACB-4000 boot payload** (§2.3)	CBIOS + `HARD.COM`'s embedded BIOS at file offset `0x053A` + the vendor's 60-byte loader block + the 14-byte hand-over stub	Available — `tools/research/hardbios.py` already reads the BIOS; the loader block is measured at `HARD.COM 0x01E5–0x0220`, and the stub is written down in `rom2image.py`
The image itself	`tools/research/rom2image.py`	**Done** — **7695 bytes**, every anchor re-checked. Built on the stock monitor it is CRC32 `28840c50`; built on the `HCBOOT`-only ROM 2.0 monitor it is `a9453f51`, and *that* is the one that boots (§4.6, §7)
The no-disk payload	the 7680-byte system image split `4096 + 3584`	**Not built** — the no-disk goal is dropped (§4.4); the split is mechanical from the above if it ever comes back

Why take the CCP/BDOS/CBIOS bytes rather than rebuild them: the CBIOS's build recipe (`bios141.gen`) names **ACT 3.5e, DDT 2.2, SYSGEN 2.1 and MOVCPM**, and ACT 3.5e is *not in the archive* (`docs/o1-rom-source.md`; 3.0 and 3.2 are, and they agree with each other on `rom144.asm`). Rebuilding the CBIOS is therefore an open risk with no payoff: the vendor's own 2048 bytes are on the disk, and the whole point of the design is that they do not need to change. The *monitor* is the part we are writing, and that is the part the toolchain is proven for.

The build therefore produces **one image**: the 4096-byte monitor plus the 3599-byte payload of §4.3 — **7695 bytes**, fitting the contiguous 8 KB window with 497 to spare. That single artefact is what the SFF's flash holds, what the interposer's flash serves to the Pico, and what the emulator sees behind the softROM device (§7). **One image, three consumers, no per-host variants.** How the payload is *presented* is measured for this boot — two contiguous 4 KB pages in the softROM's window (§4.1b) — so the emulator device models exactly that; the general window allocation is #12′s (§4.1a). The image's content is not in question.

7. The boot test (deliverable 3)

Deliverable 3 is a MAME boot test, and it is worth making it *stronger* than "no floppy attached", because "no floppy attached" is satisfiable by a ROM that hangs:

Check	Passes when
`A>` reached	the probe sees the banner and an `A>` prompt
No FDC access	the harness sees **no access to `2100–2103`** from reset to the prompt — this is the "no disk access at all" of the issue title, asserted rather than implied
The BDOS is live, without a disk	a **nonsense command** typed at the prompt is echoed and answered `?`. The CCP reads the line with BDOS function 10 and prints the reply with function 9, so this exercises the CCP⟶BDOS path end to end — and unlike `DIR`, it does not need a disk, which is the whole point
Warm boot works	a `Ctrl-C` (or a program exit) returns to `A>` **without** an FDC access
(CBIOS-ND build only) the gate holds	`DIR A:` returns `Bdos Err on A: Select` rather than hanging in the FDC retry loop — the §5-question-4 null-DPH gate doing its job

The mechanism is the one #191 already built: `tools/emulator-setup/mame-osborne1nx/` is a MAME patch that adds a machine and a device — `softrom_device`, re-registered over the ROM window (§4.1b) — the same shape as the ACB-4000 card (`docs/osborne1nx-machine.md`). The

harness is `tools/research/o1keys.sh` + `tools/research/mame-boot-probe.sh`, which is what measured every row of `docs/mame-emulation.md` §4.

Run, and it boots. The image exists — `tools/research/rom2image.py`, 7695 bytes (§6) — the softROM window serves it, and `HCBOOT` runs the chain *and* the boot: `rom2-check.sh` boots the emulated interposer **with no floppy**, presses RETURN at the sign-on prompt — and reads the dispatch's two key bytes, the CBIOS, the MD-10 BIOS, the patch, the bank switch, the hand-over stub's 14 bytes in RAM at `0E0F0h`, `CCPADR = CB00h`, the system off the drive and the `A0>` prompt back (§4.6).

Which rows that settles, exactly. The hard-disk boot is the one this project builds (§2.3), and it is measured; the no-disk payload the other three rows want is the dropped goal (§4.4), so those rows are moot rather than pending:

Check	State
`A>` reached	**measured** — `A0>` arrives with no floppy in the machine
No FDC access	**no** — the system is read off the drive, so the FDC *is* used. A boot with no disk access at all is the dropped no-disk goal, and it needs the ROM-resident CCP/BDOS, not a selector
The BDOS is live, without a disk	**no** — `0C200h` came from the drive rather than from the ROM, for the same reason
Warm boot works	**yes for the hard-disk boot** — the CBIOS's `WBOOT` reads the system off the drive exactly as the stock ROM does (§4.5)
(CBIOS-ND build only) the gate holds	**not built** — the null-DPH gate is for the no-disk build (§5 question 4)

So the ROM's own half of deliverable 3 is done and measured on the boot this project builds; the no-disk half is not a gap, it is a goal that went away.

8. What is still open

1. **The window: address, paging and selector.** For the boot this project builds it is **settled and measured**: the window is `0000–1FFF` — `mirror(0x1000)` in the machine's own address map *is* the undecoded `A12`, so a softROM that decodes it gets two contiguous 4 KB pages, with no latch and no register (§4.1b) — and `softrom_device` serves it, `-softrom` names the image, and `softrom-check.sh` accepts both. The 11776-byte payload and the paging it needed are retired with the no-disk goal (§4.4, §4.5), and the one bit that was left went too: window page 1 is the payload, always (#278). What remains open is **#12′s interface spec** for the general case — the window's address, paging and selector for the virtual peripherals — and the constraint is fixed (8 KB contiguous plus 4 KB beyond the assigned I/O block, §2.2) while the answer is not.

2. **Does ROM 2.0′s monitor code fit in 150 bytes? Yes, as it stands — and the screen is what decided it** (§4.7). `HCBOOT` is 77 bytes of the tail; with the screen's code behind it the new monitor is `103` of the tail's `149`, **46 free** to the guard. That is true because the ROM-side frame path retired (#278): with it, the screen's code was **212 bytes** against the 72 above `HCBOOT`, the assembler's guard said `CODE TOO LARGE`, and the code went to the interposer's flash as a third content id. What the fit buys is the extension table's own home: `IMSG`'s slot is 182 bytes with 110 of them free (the fallback text took 72), and `BANKSEL`, `SCRMOD`, `VERSION` and the rest are still estimates — but there is room for them, in the tail and in the vendor's dead sign-on slot.
3. **A register at `2300h`? No — measured, not inferred.** `2300–23FF` is not free: the onboard selects check two address bits each, so `0x2300` is the floppy's command/status register and `0x2301` is the floppy's track register *and* the keyboard's row 0 (`docs/o1-mainboard-schematic.md` §7a, `docs/rom-pico-protocol.md` §2). That is why the ROM↔Pico door is at `2800h`/`2801h` — the part of `docs/o1-memory-io-map.md` §7′s recommended region that nothing selects. Nothing is left to check here.
4. **Does the CBIOS really never call ROM vectors 10–12 and 16?** §3.2 says the labels do not line up and that nothing breaks; that is read from the sources, not traced. A MAME trace of a boot with a breakpoint on those addresses would settle it in an afternoon.
5. **`MEMTEST`'s scope.** Chip-level means per-4116, which needs the refresh timing and the bank-3 dim bit to be respected. Not designed here.
6. **The 512 bytes of RAM residue.** Harmless to the build (we take a fixed 2048 for the CBIOS), but it is a reminder that the *system image* is not a clean artefact and a future `SYSGEN`-equivalent should say so.
7. **The memory map at the moment the boot screen runs. Answered and measured** — `docs/rom-pico-protocol.md` §6, which also records the BIT 9 finding: the 9th video bit is a separate 4K×1 array paged into `F000–FFFF` by `OUT (02h)`, its polarity is measured from the palette, and the array is *not* a blank slate — so the screen is **two passes**, not one (§4.9).
8. **Where the frame comes from in Pico-land, and how the ROM asks for it. Answered** — `docs/rom-pico-protocol.md` §4 and §5. The animation lives in the **Pico's own flash**, compiled into the firmware, **two 4096-byte halves** (characters and attributes) per frame so the 9th bit is retained by construction, and each half carries its own trailer — which *is* the "is a frame there?" query, so there is no separate question. The ROM does not ask for a frame at all: the RP2354B plays the animation and stops on the Z80′s first video-RAM write (#278).
9. **The mailbox protocol for the ROM. Answered** — the same protocol, with **two doors, one protocol**: the I/O pair `0E7h`/`0EBh` for userland (a TPA program cannot reach bank 2) and the bank-2 register pair `2800h`/`2801h` for the ROM, which runs in bank 2 where the console path's own latch writes cannot perturb a comparator-decoded claim. `docs/rom-pico-protocol.md` §2 has the measurement that decided it. Since #278 the ROM's own client does not drive the door at all: `PRESENT`/`INFO` serve the userland client and the web UI (`firmware/rom-service/`).
10. **The hand-over: built, and the one cell was `CCPADR`.** The chain completed and the machine still did not reach `A0>`; the reason was **one cell**. The ROM's 60 Hz interrupt handler exits through `ROMJP1` — `CCPADR (0EFD2h) + 1633h` — to the CBIOS's `RRI` at `0E133h`, which is what restores the bank; the ROM's own `CBOOT` writes that cell (`02A3h`), and `HCBOOT` does not, so the exit landed on `1633h` — the softROM's page 1, the payload's copy of the CBIOS's console

dispatch — which reads IOBYTE from a page zero that in ROM mode is the ROM (D3h = the IEEE device) and never returned. Measured: the stock boot reads the IOBYTE 858 times, all 80h from RAM; the hand-over read it **once, and got D3h**. **Built.** HCBOOT copies a 14-byte stub out of the payload's spare into RAM at 0E0F0h, pushes *that* address instead of 0D800h, and SW2RAM's RET enters it; the stub writes CCPADR, the CBIOS's IOBYTE default, and jumps to 0D800h. Measured with the same instrument that found the fault: the IOBYTE now answers 80h eleven times, the guest asks for RAM 44 times in the 8 s after RETURN, and CCPADR gives E133h — and rom2-check.sh with a drive mounted reaches A0> **with no floppy in the machine** (§4.6, §7). Nothing is left on this chain; what is left in this document is #12′s window (item 1) and the extension table's contents (item 2).

Sources

- research/source-code/BIOS-144-source/rom144.asm — the monitor: the jump table (ORG 100h, lines 507–581), CBOOT (641), WBOOT (728), BCPM (746), the boot-block counts (LDK B,60 / LDK B,44), START (211), HCBOOT (4634).
- research/source-code/BIOS-144-source/occbio15.asm — the CBIOS's 34-entry jump table, MSIZE/CCP/BIOS/BDOS; occbio35.asm — ROMRI/ROMCDE/ROMCD1/ ROMJMP and the ROMVEC+3N arithmetic; occbio45.asm, occbio55.asm, occbio65.asm — HOME, the warm boot, and the two hard-coded vectors; occram15.asm — the CCP+1E00h RAM area and the size guard; occtxt6.ast — ROMVEC = 100h and the B* constants (§3.2).
- research/roms/os1-144.rom — the shipped 4096 bytes (CRC32 c0596b14).
- research/shipped-software/OS1SYSD.IMD — the DD system disk whose reserved tracks carry the CCP/BDOS/CBIOS image (§2.1).
- tools/research/romvec.py — decodes and cross-checks the jump table.
- The hand-over's exit — read from the sources, then measured: the monitor's 60 Hz handler and its ROMJP1 return (rom144.asm ~2015–2035), ROMJP1 itself (LDK DE,1633h / LD HL,CCPADR / JMP [HL], 414–421), ROMRAM = 0EF08h

  (112) and CCPADR = 0EFD2h (141, "assigned in BIOS and filled in by loader in ROM"); on the CBIOS side RRI/ROMRI (occbio15.asm 125, occbio35.asm 3–17) and SW2ROM/SW2RAM (occbio85.asm 484–506).
- tools/emulator-setup/mame-osborne1nx/rom2trace.lua, rom2bank.lua — the two instruments that measured the hand-over (#247): the PC sampler that unwinds the stack, and the taps that read the IOBYTE and count the bank selects; rom2handover.lua is the third, which proved the fix by poking CCPADR into RAM *before* it was built, and rom2boot.lua is the acceptance that checks the built store — the stub's 14 bytes in RAM, CCPADR, and the prompt.
- tools/research/rom144-reassemble.sh, tools/research/acthex.py — the toolchain proof (#203). acthex.py's --fill-reference and --covered-range are what let a build whose hex file the emulator's floppy write-back damaged still be the image the source describes, and stop the build loudly when it cannot (docs/rom-13-reconstruction.md §7).

- `docs/o1-rom-source.md`, `docs/o1-rom-variants.md`, `docs/o1-memory-io-map.md`, `docs/mame-emulation.md`, `docs/osborne1nx-machine.md`, `docs/ozrom-1e.md`, `docs/md10-system.md`, `docs/acb4000-host-adapter.md`.

Video and the character generator

Status: Partially solved. The set is identified and read straight off the ROM — 33 glyphs, and they are **block graphics, not line-drawing** — and the "64 columns" question is answered by the video geometry. The glyph-by-glyph mapping to Unicode codepoints is deliberately **not** written yet: it has to be measured glyph by glyph, not asserted (see *The mapping (open)*).

The 2716 at **UA15** holds 128 glyphs of 8×10, and the first 32 of them (`0x00–0x1F`) plus `0x7F` are **graphics characters** rather than letters. They are the most stable thing in the ROM: the two char-gen variants differ in 93 of 128 glyphs, and the `0x00–0x1F` set is byte-identical in both (`docs/o1-rom-variants.md`).

They are block graphics, not box-drawing

The natural assumption is the IBM/CP437 line-drawing set (─│┌┐└┘├┤┬┴┼). That assumption is wrong, and the manual says so in its own section title:

> **5.5 BLOCK GRAPHICS** — *Osborne 1 Technical Manual* p. 25

Looking at them confirms it — the sheet is committed beside this file as `o1-chargen-graphics.png`, rendered by `tools/romfont.py --glyph-sheet`:

- **halves and quadrants** (solid blocks filling half or a quarter of the cell)
- **staircases and triangles** (the diagonal edges, drawn in steps because an 8×10 cell has no other way to do a diagonal)
- a **circle** and a **cross**
- `0x7F`, a **checkerboard/hatch** fill — the glyph the site's wordmark is hatched with

So the candidates to compare against are **Block Elements** (`U+2580–U+259F`) and Geometric Shapes, not the box-drawing block. That changes the mapping work from "guess a line junction" to "measure a fraction of the cell", which is a much better position: a block glyph can be *derived* from its coverage.

What the set is for

It is what the machine used for graphics on a character display — the same purpose as CP437's block and box characters, and the reason the O1 can draw frames and charts at all. Two consequences for this project:

- **A video personality (#24) has to render these correctly**, not just the letters. `docs/o1-memory-io-map.md` §2a has the cell and attribute layout.
- **They are page furniture (#97).** A period site's rules, table borders, footer and bullets can be the machine's own characters instead of CSS borders.

The mapping (open)
There is no codepoint table yet, and that is a method decision rather than a gap: each of the 33 glyphs is identified by looking at it beside its candidate and recording it with that evidence, not asserted from a description. The sheet is the evidence side of that work — `tools/romfont.py --glyph-sheet` renders it from the ROM, so it cannot drift from the ROM.

Next: render a Unicode Block Elements chart beside the sheet, identify each of the 33, then add the table to `tools/romfont.py` so the codepoints ship in the font.

Why 64 columns (see also §2a)
The recovered AUTOCPM art is 64 columns wide and the O1′s screen shows 52. That is not a contradiction, and the SCREEN-PAC guide supplies the missing piece: the machine separates **logical** width from **physical** width.

> Logical screen size refers to the number of characters you can enter on a line without automatically inserting a carriage return. Logical screen size can range from 1 up to a **maximum of 128 characters**. … Physical screen size refers to the number of columns actually displayed on your screen. With SCREEN-PAC, this will be 52, 80, or 104. — *Osborne 1 SCREEN-PAC User Guide*, p. 13

A 64-column drawing is a **logical line of 64 characters**: it fits the 128-character video memory row, and the physical window slides over it. On stock hardware you see 52 of the 64 columns at once and scroll for the rest; with the SCREEN-PAC at 80 you see all of it.

The full statement, with the technical manual's geometry and the attribute bits, is in `docs/o1-memory-io-map.md` §2a.

Sources
- `research/osborne1/2F00153-01_Osborne1TechnicalManual_1982.pdf` §5.0–5.5, pp. 23–25 (image-only scan; read by rendering the page range — the manual's page 15 is PDF page 29, so the offset is **+14**).
- `tools/romfont.py` — the decoder and the `--glyph-sheet` renderer.
- `docs/o1-rom-variants.md` — the char-gen variants and what differs.

Interfaces

The floppy subsystem and its formats

Status: Solved. The mechanism is proven from the 1.44 CBIOS source (`occbio95.asm`) and exercised end to end in MAME on an Osborne **single-density** disk in a double-density system — the recognition step itself; the four foreign formats are **taken as written**, not synthesised, because the archive holds no media for them and the O1 never has to recognise them (#95).

The DD upgrade is usually described as "twice the capacity". Its own manual describes something else: **a controller that recognises six disk formats**, five of them other manufacturers', with the *computer* doing the recognising. The six, at the manual's own figures, are **Osborne 1 single den-**

sity (92 k), **Osborne 1 double density (182 k)**, **Xerox 820 single density (80 k)**, **Cromemco single density (80 k)**, **IBM Personal Computer, CP/M-86 format (156 k)** and **DEC VT-180 (171 k)**. Six names, five disk parameter blocks — Cromemco shares the Xerox geometry — and the mechanism is two cooperating parts: the monitor ROM *senses* the disk by trying both densities and reading an address mark, and the DD CBIOS *matches* the sensed geometry against a DPB + sector-skew table (§"Why the ROM is part of the answer"). That table's five rows reproduce the manual's capacities field for field, which is what makes this the answer rather than a guess.

The list below is the manual's, and it is the list to follow: where a secondary source differs — oldcomputers.net gives the Xerox 820 entry as **82K** — the manufacturer's own page-4 statement of what the option does wins.

> Thus, you'll find that you didn't just purchase double density, you purchased a **multi-density disk option**. Under the current release of double density, the following formats are all **automatically recognized by the computer**.
>
> | Format | Capacity |
> |---|---|
> | Osborne 1 single density | 92k per diskette |
> | Osborne 1 double density | 182k per diskette |
> | Xerox 820 single density | **80k** per diskette |
> | Cromemco single density | 80k per diskette |
> | IBM Personal Computer (CP/M-86 format) | 156k per diskette |
> | DEC VT-180 | 171k per diskette |
>
> — *Osborne 1 Double Density User Manual*, p. 4

That is the primary source for #95. Note the wording — "automatically recognized by the computer": the recognition is in the machine, which is what makes the claim testable in MAME without the hardware.

The manual also records what the option deliberately does **not** handle, which matters as much as what it does:

- **Hard-sectored diskettes** (North Star, Micropolis, Vector) — the controller cannot see the sector holes.
- **Non-block formats** (Apple, NEC) — no block-of-bytes structure to read.

and names the formats that were *anticipated* rather than shipped: TRS-80 Model I and III CP/M, and Zenith/Heath double density, "most likely in our user magazine, The Portable Companion".

Why the ROM is part of the answer

The upgrade is described as coming with the **1.4 ROM**, and the recognition is a property of the computer rather than of the drive electronics. The 1.44 BIOS source and the overlay CBIOS modules are in the repo (`research/source-code/BIOS-144-source/`), and reading them settles #95 without hardware. The mechanism is **two cooperating parts, and neither is the monitor ROM alone**:

1. **The monitor ROM senses the disk.** `SENDEN` (jump-table entry `0130`, `SENSE_DISK_DENSITY`) tries both densities and issues a WD179x **READ ADDRESS** (`RADR`) on track 0. The address mark the controller finds *is* the disk's geometry: it yields the density (FM/MFM), the sector length and the sectors per track. There is no format table in the monitor ROM — the geometry comes off the disk. `SAVTYP` is set to a code of `density (bit 0) | sector length (bits 2–3)`, where the length field is `0`=128, `1`=256, `2`=512, `3`=1024 bytes.
2. **The DD CBIOS carries the DPB table.** `occbio95.asm` defines a table of Disk Parameter Blocks, and `occbio45.asm` (`GDEN`) converts the sensed physical sector count into a count of 128-byte *logical* sectors (×1/×2/×4/×8 for 128/256/512/1024) and searches the table for the entry whose sectors-per-track **and** type code both match. On a match it installs that DPB and its sector-skew translation table.

The table is the smoking gun (`occbio95.asm` / `occbio15.asm`, the names are the manufacturer's own):

Entry	Type code	Sectors/ track	Block	Dir	boottrk	`DSM` formula	**capacity**	Manual's figure
Osborne DD (`DSKD1`)	`0Ch` = MFM, 1024 B	40 (=8×5)	1 K	64	3	`((40-3)*8*5)/8` = 185	185 K	182 k
Osborne SD (`DSKS1`)	`5` = FM, 256 B	20 (=2×10)	2 K	64	3	`((40-3)*2*10)/4` = 46	46 blk × 2 K = 92 K	92 k
Xerox (`XEROX`)	`1` = FM, 128 B	18 (=1×18)	1 K	32	3	`((40-3)*1*18)/8` = 83	83 K	80 k
IBM (`IBM`)	`8` = MFM, 512 B	32 (=4×8)	1 K	64	1	`((40-1)*4*8)/8` = 156	156 K	156 k
DEC (`DEC`)	`8` = MFM, 512 B	36 (=4×9)	1 K	64	2	`((40-2)*4*9)/8` = 171	171 K	171 k

`NUMDPB = 5`. **The capacities match the manual's list exactly** — IBM 156 k and DEC 171 k to the kilobyte — which is what makes this the answer rather than a guess. The manual lists **six** formats and the table has **five** entries because **Cromemco shares the Xerox geometry** (both 5.25" single-density, 80 k), so one DPB serves both names.

Each entry also carries a **sector-skew translation table** (`XTAB` in `occbio95.asm`): Osborne DD is 1:1, Osborne SD is 2-to-1, Xerox is 5-to-1, IBM is 1:1, DEC is 2-to-1. "Mostly DPB + FDC parameter tables", as #30's text put it — confirmed, and the *skew* is the part that would be missed by a geometry-only description.

So the DD upgrade's multi-format claim is a **CBIOS DPB+skew table selected by a monitor-ROM density/sector probe** — recognition in the computer, exactly as the manual says, with the *filesystem* parameters living on the diskette's own CBIOS rather than in the ROM. That also explains the G2 answer (#30): a "G2 format set" is more entries in the same table.

What this means for the project

Decision (2026-10-11): take the manual as written, and do not synthesise the four foreign formats. The claim has two independent sources that agree — the manufacturer's own page-4 list, and the CBIOS DPB + skew table whose five rows reproduce the manual's capacities field for field — and the mechanism is exercised end to end in MAME on the one format we hold media for (Osborne SD, in the test below). What a four-format test would add is the four **skew** entries and nothing else: a capacity cannot catch a wrong skew, only a `DIR` can. Those are skews for formats we are choosing not to serve.

- **The floppy personalities (#20, #62) must not *assume* Osborne geometry — but they do not have to implement six formats either.** A Gotek-class emulator is format-agnostic: an `.HFE` carries the track layout, so replaying it is not the same thing as knowing the format. Format knowledge is only needed to *synthesise* a blank foreign diskette, and that is out of scope.
- **Foreign media is converted on the modern side, before it is mounted.** The O1 never has to recognise anything: read the files out of a foreign image with `tools/cpmfs.py` (it already reads these geometries) and write them into an Osborne-format image with `mkfs.cpm` + `cpmcp -T raw`. That is why the four skew tables do not need verifying.
- **The G2 (#30) is more rows in the same table — and no rows get written.** #30 asked for a G2 format catalogue; the fold into #95 carried the *mechanism*, not the catalogue, and with the decision above the catalogue is not needed either.

A second vendor made the same claim, and a bigger one: Nuevo

OCC is not the only multi-format claim of the period. Nuevo's four-way product line, advertised in *The Portable Companion* #15 (Jan/Feb 1985, PDF p. 39), escalates from density to format:

Nuevo product	Price	Claim
Double Density Upgrade	$150	"a full **183K** of data storage per disk" — and *"upgradable to double-sided double density"*
2 x 2 Upgrade (780K)	$595	double-sided double density, "nearly **400K each**", *"Format compatible with the new ACC Vixen"*, **128 directory entries**
4 x 4 Upgrade (1580K)	$745	"both drives are replaced with **double-sided 96 track-per-inch drives**"

The 4 x 4 text is the important one:

> This system will **automatically recognize single-sided disks that are either single or double density and double-sided disks that are either 48 TPI or 96 TPI**. You can **read or write, even boot**, any of these discs automatically. Directory Capacity 256 File Names.

So Nuevo claims four formats beyond the six in OCC's DD table — 96 TPI, both densities, both sidednesses — and claims to **boot** them. Two things make that credible rather than just advertising:

- **The stock O1 interface already carries what it needs.** P8 pin 4 is **TG43**, the drive's write-current-reduce input for 96 tpi media, and pin 32 is **SIDE SELECT** — both documented in `docs/floppy-adapter-design.md` from the schematic and the OEM cable. The controller is not the limit; the drives and the format table are.
- **The format table is the CBIOS's job, and Nuevo replaces it.** The installers on `OS1NUEVO.IMD` are DDT patches inserting `BIOS15.OBJ` at `1F00` and `BIOS15BM.OBJ` at `29E0` (`docs/o1-upgrade-boards.md`), and the advertised directory capacities — 128 and 256 entries — are disk parameter block values, exactly what such a patch changes. The 2 x 2 and 4 x 4 prices include **new drives**, so the media capability is hardware and the *recognition* is the BIOS's.

Consequence for #95 and #30: the multi-format requirement is not an OCC quirk. Two vendors shipped it, with overlapping but different format lists, and the O1′s own floppy connector carries the signals for both. That argues for a floppy personality (#20) that is **geometry-agnostic** — replay what the image says — rather than one that hard-codes a single format. It does *not* argue for implementing every format: the images we mount are converted to native geometries first, as the decision above records.

Nuevo's own BIOS 1.4 → 1.5 list: the formats are disk parameter blocks
Nuevo's DD manual has a passage that settles the mechanism behind all of this. Its BIOS 1.5 differs from the factory BIOS 1.4 in ways it enumerates itself:

> 2) For BIOS Version 1.5 the **disk parameter blocks for IBM, Xerox and DEC formats have been removed** to provide space for code to allow **auto configuration at boot time via NEWSETUP.COM**. Space remains for two more DPB's to be inserted.
>
> 3) BIOS Version 1.5 **checks for the presence of the Nuevo 80 Column Upgrade**. If not present, the configuration routine for the 80 Column Upgrade will be skipped. Thus, the Version 1.5 BIOS will boot and run correctly on machines that don't have the Nuevo 80 Column Upgrade.
>
> 4) The jump table configuration and the jump table start address (**E100h**) is the same in BIOS Version 1.5 as in BIOS Version 1.4. — *Nuevo Double Density Upgrade Manual*, "differences between the two BIOS versions" (OCR: `research/nuevo/Nuevo_Electronics_DD_Upgrade_Manual.ocr.txt`)

Five things follow:

- **"BIOS 1.4" and "BIOS 1.5" here are the *CBIOS on the system tracks*, not the monitor ROM.** Nuevo's installers load `BIOS15.OBJ` at `1F00` and `BIOS15BM.OBJ` at `29E0` (`docs/o1-upgrade-boards.md`), and the manual compares 1.5 against *"Version 1.4, the most recent **factory version**"* — so the DPB table it describes lives in the bootable CBIOS image, and *that* is the file to read for #95, not the 4 KiB monitor ROM.
- **The monitor ROM contains no literal DPBs, and the factory BIOS does not need them.** A structural scan of all four ROMs the project holds — factory 1.4, 1.43, 1.44 and Nuevo 1.51 — finds **zero** CP/M DPB-shaped structures. The reason is in the 1.44 source: the loader *"will load single or double density and any number of sectors per track or bytes per sector"*, and `SENSE THE DENSITY` **probes** — it reads an address mark at one density, flips the density bit, and retries, with a second pass after a home command (`rom144.asm` lines 3666–3712). Geometry is discovered by trying, not by table lookup.
- **Nuevo is not a superset of the factory BIOS.** Nuevo *removed* IBM, Xerox and DEC to make room for its own code and re-added formats through **auto-configuration at boot via NEWSETUP.COM**. Two vendors, two mechanisms, overlapping lists — the same conclusion as above, now with the reason.
- **"Space remains for two more DPB's to be inserted"** — the table is extensible, which is how the 2 x 2 and 4 x 4 geometries fit into the same scheme.
- **Nuevo's BIOS checks for its own video board's presence** and skips the 80-column configuration if it is absent — the same *intent* as the SCREEN-PAC's `2400H` flag, reached by a different mechanism (`docs/o1-upgrade-boards.md` §"How software finds the board"), and the reason its 1.5 ROM still boots on a machine with no Nuevo hardware. The jump table sits at **E100h**, unchanged from 1.4, which is the region the SCREEN-PAC's width variable (`E1A5H`, `docs/o1-memory-io-map.md` §2a) lives in.

The 1.4 → 1.5 list, checked against two images (#173)

The list above is the vendor's. It can now be checked, because the archive holds both sides: `research/shipped-software/O1CPM.IMD` is a factory DD boot disk whose system image is the same 8 KB CP/M build as Nuevo's base, and `research/nuevo/OS1NUEVO.IMD` carries CBIOS 1.5 both as the installer's DDT objects and as the system it boots. Five of the six items are confirmed in the bytes:

Manual item	Where it shows
1 — the `59k` sign-on removed	`Osborne Computer System` / `59k CP/M vers 2.2` becomes `Osborne 1 Computer` / `CP/M vers 2.2` / `CBIOS vers 1.5`
2 — IBM/Xerox/DEC DPBs removed	at system offset `172C` the factory carries three sector-translation tables and Nuevo keeps the first and **zeros the other two**, with a pointer table (`E236`, `E24A`, `E93C`, `E95C`) where the first table's header was
3 — autostart names 7 → 8 characters	the field **moves**: 1.4 keeps it at `CB10–CB17` (8 spaces, unused), 1.5 at `CB87–CB90` (`06` `"AUTOST"` `00` + padding)
5 — the "Unformatted disk" message deleted	1.4 selects between two messages on bit 0 and patches the drive letter into **both** (`E38C`); 1.5 has one message and one patch site (`E340`)
6 — jump table at `E100h`, unchanged	19 `JP` entries at `E100` in both images; every target moved
4 — the presence check	not a register: `IN A,(04H)` at the head of the cold boot, guarding the 80-column configuration — `docs/o1-upgrade-boards.md` §"How software finds the board"

Item 3 is worth its own line, because it explains an installer step that otherwise looks arbitrary: `SYS-80C.SUB`'s `FA10,A17,00` is not "clearing the width/variable block", it is **erasing the 1.4 autostart field** — 1.5 keeps the name somewhere else, and `0010–0017` of the image is exactly the 8 bytes the fill clears.

A third vendor sold the format list itself: Osmosis OSMOS 4

Osmosis advertised the capability as a **standalone product**, in *The Portable Companion* #11 (October 1983), PDF p. 32 = printed p. 30:

> **OSMOS 4** — Disk Formatting & Conversion: Allowing your Osborne to **read, write and format over 20 other manufactures CP/M formats**. $225

That is the strongest form of the claim in the archive: not a capability bundled into a density upgrade (OCC), and not a side effect of a BIOS rewrite (Nuevo), but a **$225 product sold on its own**, priced *above* the $195 double-density board it was usually bought with. The same page bundles them — "**OSMOS 1, 4 AND 5 ALL FOR $385**", $670 separately — which is a fair measure of how much of the value was seen to be in *formats* rather than in *density*.

Its density line, for the record: **OSMOS 1** 184k per drive on existing drives ($195), **OSMOS 1.5** single-sided DD now, 370k later with new drives ($335), **OSMOS 2** double-sided DD, 370k ($895), **OSMOS 3** double-sided DD **double tracking**, 750k per drive, "an incredible 1.5MB" ($1050). The full product table is in `docs/o1-upgrade-boards.md`.

The tally is four independent format mechanisms: OCC's DPB table, Nuevo's boot-time auto-configuration, Osmosis's separate conversion program, and the factory BIOS's density/sector probing. A floppy personality (#20) that knows only the stock geometry fails on any of them.

What the DD upgrade costs in memory: the Corvus link program

A user note in *The Portable Companion* #11 (October 1983), PDF p. 54 = printed p. 52 — "**Doubling hard disk memory**" by Bob Gras, reprinted from the OSUG newsletter (Anchorage) — is a patch to Corvus's link program, and it says plainly why it is needed:

> If you had a single-density Osborne computer, but now have a double-density and are using the Corvus hard disk drives, this one is for you. … With the new double density system, **more memory is required for managing the Osborne floppy drives**, so this value has to be reset to accommodate the double density version of the Osborne.

The value is a CP/M offset in `CLINK2K2.ASM` (the source for the `CLINK2` program that links the machine to the Corvus drive):

```
IF NOT (ABOVE)      ;
DELTA  EQU  500H    ; OFFSET FROM STANDARD CP/M *
```

and the fix is to "type a **900H** in place of the **500H**", then assemble and link with `ASM.COM` and `LOAD.COM` (the article points at the Osborne manual's assembly instructions, pp. 253–255 of the "Rev: 2/28/82" edition). So the constant grows by `400H` — 1 KB — between the single- and double-density versions of the same machine, and the author's stated reason is the DD system's larger floppy- management code. **The DD upgrade is not memory-neutral**, which matters for anything that assumes a fixed TPA base, and it is the kind of third-party consequence the DD documentation itself does not mention (#95).

The acceptance test, run in MAME (#95)

The manual's claim is a capability, and the capability is testable without hardware because the recognition is in the computer. The test: **boot ROM 1.4+ with a DD CBIOS, put a disk in drive B, move files on and off it, and read the result back byte-for-byte.**

Tooling

Two small stdlib helpers were added for this (`tools/research/`):

- **`imdmake.py`** — writes a blank Osborne 1 disk in ImageDisk form (the writer half of `imdinfo.py`): `--dd` = 250 kbit MFM, 40×5×1024; `--sd` = 250 kbit FM, 40×10×256, filled with `0xE5`. It emits the `IMD 1.x:` version line first, because `disk-analyse` refuses a file that does not start with one.
- **`o1disk-copy.sh` + `o1disk-copy.lua`** — boot a disk in `-flop1`, put another in `-flop2`, wait for `A>`, type a CP/M command, and report the screen. It is the keyboard-injecting counterpart to `mame-boot-probe.sh`.

Results (measured)

Test	Command	Result
ROM 1.4+ boots DD	`mame osborne1 -bios ver14/ver143/ver144` on `52-blank.imd`	**all three boot to `A>`**
Content onto B:	`PIP B:=A:MBASIC.COM`	B: image gains `MBASIC.COM`, **byte-identical** to A:'s (24320 B)
Content off B:	`PIP A:BACK.COM=B:MBASIC.COM`	A: image gains `BACK.COM`, **byte-identical** — full round trip
Foreign density read	DD system in A:, **Osborne SD** disk in B:, `DIR B:`	**lists B:'s files** — the DD CBIOS auto-recognised the single-density disk

The last row is the important one for this issue: it exercises the real mechanism end to end. The monitor ROM's `SENDEN` read the SD disk's address marks (FM, 256-byte sectors, 20/track), the DD CBIOS matched `DPBS1`, and `DIR` read a foreign-density directory. **Osborne SD is one of the manual's six formats, confirmed on the emulated machine.**

A MAME gotcha worth writing down

`os.exit(0)` in the Lua script silently discards guest disk writes. MAME flushes a modified image when the device is torn down at a *clean* shutdown; `os.exit()` skips that. A `SAVE 1 HELLO.TXT` under `os.exit()` left the image byte-identical; the same command with the script simply ending (MAME running on to `-seconds_to_run`) persisted. `o1disk-copy.lua` therefore does **not** call `os.exit()` — the opposite of `mame-boot-probe.lua`, which only reads.

What the test does *not* yet cover

- **The other four formats** (Xerox 820, Cromemco, IBM CP/M-86, DEC VT-180) — we hold no images for them, so the DPB-table proof above stands on the source rather than on a run. The way to close it is to synthesise an image per DPB entry with `imdmake.py` (or `disk-analyse`) and `DIR` it; that is the next unit of work, and it is cheap now that the geometries are known.
- **Real hardware** — see below.

IMD → HFE for the real machine

`disk-analyse` converts both ways, and the round trip is lossless on the raw sector data:

```
disk-analyse blank-dd.imd blank-dd.hfe      # write to a Gotek/HxC
disk-analyse blank-dd.hfe blank-dd-back.imd # read it back
tools/research/imdinfo.py --raw a.raw blank-dd.imd
tools/research/imdinfo.py --raw b.raw blank-dd-back.imd
cmp a.raw b.raw                             # identical
```

Measured: a `--dd` blank and the `52-blank.imd` system disk both convert to HFE (2 039 296 bytes each), and IMD→HFE→IMD reproduces the raw sector image exactly. `disk-analyse`'s HFE output is what a hardware floppy emulator wants, so the images built and validated above are directly testable on a real Osborne 1.

Sweep of the archive (measured 2026-10-11). Every `.imd` we hold converts — **22 of 22, no failures** — including the ones that are *not* Osborne-format: the double-density system disks, all fourteen Dunfield shipped-software images (SD and DD), `DRIVE_C.IMD`, `OS1NUEVO.IMD`, `OCC1_HARDDISK.IMD` and the two diagnostic disks. A spot round trip on the single-density 80-column `OS1XUTLS.IMD` reproduces the raw sector image byte-for-byte. So the "foreign" images in the collection are ready to write to a hardware emulator.

```
for f in research/shipped-software/*.IMD research/drive_c/DRIVE_C.IMD; do
    disk-analyse "$f" "$(basename "${f%.*}.hfe")"
done
```

Sources

- `research/source-code/BIOS-144-source/` — `occbio95.asm` (the DPB + skew table), `occbio45.asm` (GDEN), `rom144.asm` (the density probe, SENDEN), `occram25.asm`.
- `research/nuevo/Nuevo_Electronics_DD_Upgrade_Manual.pdf` (+ `.ocr.txt`) and `research/nuevo/OS1NUEVO.IMD` — Nuevo's BIOS 1.4 → 1.5 list, its CBIOS 1.5 and its installers.
- `research/shipped-software/*.IMD` — the disk images the HFE sweep and the MAME acceptance test run on; `O1CPM.IMD` is the factory DD boot disk used as the CBIOS 1.4 diff partner.
- `research/osborne1/o1doubledensityusersguide.pdf` — *Osborne 1 Double Density User Manual* (15 pages, image-only; OCR'd per the convention in `research/README.md` as `o1doubledensityusersguide.ocr.txt`). https://www.kev.pulo.com.au/osborne1/documents/o1doubledensityusersguide.pdf
- `docs/o1-memory-io-map.md` §3 (WD179x detail) — the formats already recorded there from the schematic side.
- `docs/o1-upgrade-boards.md` §"How software finds the board" — the 80-column half of the same vendors' product lines.

The floppy adapter design

Status: Solved as a design. Every decision this design needed is made (§8), the Floppy Adapter's Rev A schematic is captured, wired, footprinted and checked, and the board is ready for PCB floorplan and routing; nothing is built and nothing is measured on hardware.

Design spec for issue **#62**. Tracks the floppy interposer personality (#20); the disk-format position is settled in `docs/dd-upgrade-formats.md` (#95). Derived from `research/osborne1/floppy-adapter/` (Wayne Visser passive adapter + provenance), Richard Loxley's Osborne Restoration parts 5 and 12–15, and `hardware/floppy-adapter/README.md`.

The switch is solid state — three silicon parts — decided 2026-10-22. The case has no height for relay blocks, so DS-A is switched by an **NC7SB3157P6X**, the Gotek's +5 V by a **TPS22918** and its ground return by an **AO3400A** (§4.3, §4.4). The fail-safe survives: the switch's own truth table parks the machine on the physical drive. Rev A KiCad capture lives in `hardware/floppy-adapter/` (`floppy-adapter.kicad_sch`, generated by `gen_floppy_adapter_sch.py`).

> **Vocabulary:** `Rev A` / `Rev B` in this document are the **Floppy Adapter's** revisions. That is an unrelated board to the interposer's Rev A / Rev B, so the letters always carry this board's name — see `hardware/README.md`.

1. Purpose

Let the user choose, from a front-panel switch, whether the **drive A** position is served by the physical drive or by a **Gotek** (FlashFloppy) emulator, with **drive B always present** in both modes. The card sits between the O1 mainboard/DD-board floppy header and the OEM floppy cable; nothing in the machine or its cable is cut.

Prior art, deliberately **not** copied:

- Wayne Visser's `Osborne1FloppyAdapter` is a **passive** tap — it puts a Gotek on the bus alongside the physical drives and switches nothing. We keep its pin/rail mapping as the O1 interface reference.
- Richard Loxley's hand-wired build (parts 12–15) proved the concept but needed a DPDT switch + a 5 V relay soldered onto the Gotek, plus hand-cut Dupont cable. This design moves all of that onto a PCB at the connector end of the cable.

2. Osborne 1 floppy interface (34-pin, Shugart-like)

The O1 uses a Shugart-like 34-pin interface, but with **+5 V and +12 V routed to the drives over otherwise-unused cable pins** — there are no separate drive power connectors. The pin functions below are **read from the O1 mainboard schematic itself** — DWG 1A2011-00 Rev E, sheet 9 of 9 (DISC CONTROLLER), the P8 symbol; the same pinout is extracted in `docs/o1-mainboard-schematic.md` §5 — and they agree with the archived Wayne Visser adapter (`research/osborne1/floppy-adapter/FloppyAdapter.sch`), Loxley's signal audit and the standard Shugart ordering for the shared signals.

Pin(s)	Signal	Notes
1,2,3,5,9,19,27,29,31,33	GND	pin 7 is GND on the drive-end drawing instead; neither 2 nor 7 is a driven signal, so both are tied to GND here
4	**TG43**	**not GND** — FDC-driven; the drive's write-current-reduce input on 96 tpi drives
6	**EARLY**	**not GND** — FDC-driven
7	GND (drive-end drawing)	see note on pin 1 row
8	INDEX	
10	DS-A (drive select A)	**the switched line** — Gotek replaces drive A
11,13,15,17	+12 V	power-over-cable (physical drives)
12	DS-B (drive select B)	not switched — drive B stays present
14	*(not drawn on P8)*	
16	4 MHz	O1-specific clock; not used by Gotek
18	DIRECTION	
20	STEP	
21,23,25	+5 V	power-over-cable (physical drives; tapped for Gotek)
22	WRITE DATA	
24	WRITE GATE	
26	TRACK 0	
28	WRITE PROTECT	
30	READ DATA	
32	SIDE SELECT	
34	LATE	O1-specific

Pins 4 and 6 carry live FDC outputs, so the card **passes them straight through J1 → J2** rather than grounding them. This card switches the OEM cable through to the physical drives, and clamping TG43 low would assert reduced write current permanently; a Gotek-only adapter that grounds 2, 4, 6 and 7 does not have that problem, which is the whole difference between the two.

The Gotek needs only the static/step/data lines plus its drive-select (Loxley's 11-line audit). The card routes that subset to the Gotek header and leaves 4 MHz and the spare power pins off it.

3. Topology

```
flowchart LR
  MB["O1 mainboard<br/>34-pin floppy header"]
  subgraph CARD["floppy-adapter daughter card"]
    PLUG["J1  2x17 socket (bottom)"]
    CABLE["J2  2x17 header (top)<br/>to OEM cable"]
    GOT["J3  2x17 header (top)<br/>to Gotek signals"]
    PWR["J4  Gotek power<br/>(switched +5V, GND)"]
    SW["J5  panel-switch lead"]
    U1["U1  2:1 switch<br/>(DS-A)"]
    U2["U2  load switch<br/>(Gotek +5V)"]
    Q1["Q1  N-MOSFET<br/>(Gotek GND)"]
  end
  DA["Physical drive A<br/>(150 ohm term pack)"]
  DB["Physical drive B<br/>(always present)"]
  GK["Gotek / FlashFloppy<br/>(as drive A)"]

  MB -- "J1 mates header" --> PLUG
  PLUG -- "pass-through (all pins)" --> CABLE
  CABLE --> DA
  CABLE --> DB
  PLUG -- "signals only" --> GOT
  GOT --> GK
  U2 -- "+5V (when Gotek)" --> PWR
  Q1 -- "GND (when Gotek)" --> PWR
  PWR --> GK
  SW -- "logic only" --> U1
  SW -- "logic only" --> U2
  SW -- "logic only" --> Q1
  PLUG -. "DS-A (pin 10) in" .-> U1
  U1 -. "DS-A to physical A (S low)" .-> CABLE
  U1 -. "DS-A to Gotek (S high)" .-> GOT
```

Everything between J1 and J2 is a **straight pass-through** except pin 10 (DS-A), which is intercepted. The physical drives therefore see an unmodified cable; the only difference is which device the machine's DS-A reaches.

4. Switching design

4.1 Drive-select routing (pin 10)

DS-A from the mainboard (J1.10) is U1′s common. **Control low** connects it to J2.10 ⟶ the physical A drive; **control high** connects it to J3.10 ⟶ the Gotek. Never both — and U1′s break-before-make guarantees that even during the changeover — so the two devices cannot both answer as drive A. DS-B (pin 12) is untouched, so drive B stays present and selectable in both modes.

4.2 Gotek power switching

Loxley's field notes (part 15) are decisive and are the reason this card exists:

- **Switching only +5 V does not turn the Gotek off** — the O1′s pulled-up signal lines parasitically power the board through its input protection diodes.
- **Switching only GND makes the *other* drive unreliable** — the Gotek's signal lines are still referenced to the shared 5 V.
- **Switching both +5 V and GND works.** That is what this card does.

Consequence: the Gotek's **GND and +5 V must arrive only through J4**, *not* through the J3 header, otherwise switching GND has no effect. J3 carries signals only.

4.3 Switch element — DECIDED 2026-10-22: solid state, because there is no height for a relay

The decision, and the constraint that decides it. On **2026-10-22** the maintainer settled the switch element as solid state, with a constraint this card cannot argue with:

> **There is not enough height in the case for relay blocks.**

That is decisive, and it is not about price or availability. A DIP-14 signal relay is ~10 mm tall; an industrial 4PDT on a socket is 35 mm. The card is a **mezzanine between the logic board and the floppy harness**, and the machine has no room for either. **Rev A is solid state** — three SMD packages, ~1.5 mm.

The distribution check agrees, independently: there is no 4PDT DIP signal relay in production to design around. Omron **G6A-434P** / **G6A-474P** were discontinued with a last-buy date of 2010-06-29, and the 4-pole DIP class went with them — Panasonic **TQ4-5V** and **DS4E-S-DC5V** are broker-only at **$20–186** a piece.

	relay — the option rejected	solid state — decided (Rev A)
height	~10 mm DIP-14; 35 mm socketed	**three SMD packages, ~1.5 mm**
what the panel lead carries	56–73 mA of coil current	**microamps**
DS-A	a contact pair	**NC7SB3157P6X**, ≤10 Ω, break-before-make
the two rails	two more contact pairs	**TPS22918** (52 mΩ) + **AO3400A** (28 mΩ)
fail-safe	de-energised → NC contact	**S low → B0 → the physical drive**
parts	1 (unobtainable) or 2	3 + 4 passives
cost	$4–12 for two DPDT	**≈ $3**
click, and contact life	yes; 10^5–10^7 operations	none; no wear-out

4.4 The three switches

Three parts, one control node (`CTRL`), and the same fail-safe the relay's NC contact gave:

Ref	Part	What it does
U1	**NC7SB3157P6X** — single 2:1 analog/bus switch, SC70-6	picks which device owns DS-A. `A` is the machine's DS-A; **`B0` is the physical drive**, `B1` is the Gotek
U2	**TPS22918** — 5.5 V / 2 A load switch, 52 mΩ, SOT-23-6	switches the Gotek's **+5 V**; `VIN` is the cable's 5 V, `VOUT` is `+5V_SW`
Q1	**AO3400A** — N-MOSFET, 28 mΩ at Vgs = 4.5 V, SOT-23	switches the Gotek's **GND** return (`GND_SW`)

The truth table is the fail-safe, and it is the part's own. The NC7SB3157's datasheet says *S low → B0 connected to A; S high → B1 connected to A*, so wiring B0 to the physical drive means a control that is low, floating, or has lost its lead parks the machine on the real drive. R1 (10 kΩ) pulls `CTRL` down for the same reason the relay design pulled the coil node down. The part also has **break-before-make** select circuitry, so the two devices are never both connected to DS-A even for an instant — the property the relay gave for free, and the one that must not be lost in a solid-state version.

Why these parts, and not the others.

- **A bus switch on DS-A, not a 74HC4053 or CD4053.** The DS-A line is pulled up by 1 kΩ on the Gotek side; a 70–120 Ω switch in series with that puts the low level at 0.7–0.9 V against a

0.8 V TTL threshold — a part that works on paper and fails on a machine. The NC7SB3157 is a 5 V rail-to-rail part at ≤10 Ω, and its control input tolerates 5 V independent of VCC.

- **A MOSFET on the ground return, not a second PhotoMOS.** An AQY212GS is 0.7 Ω, which at the Gotek's ~0.3–0.5 A is 0.2–0.35 V of ground offset between the Gotek and the machine's TTL signals — half the noise margin, on the read-data line. The AO3400A is 28 mΩ, which is **14 mV** at half an amp. (Two AQY212GS *would* have been the simplest possible build — two parts, no 5 V-level care — and this is the number that rules it out.)
- **A load switch on +5 V, not a discrete P-FET pair.** The TPS22918 is one part with a logic-level ON pin, 52 mΩ, and a configurable rise time; `CT` and `QOD` are left open (open `CT` is the fastest rise, open `QOD` disables the discharge). If the bench ever shows the Gotek lingering on power-down, `QOD` is the pin to come back to.
- **No flyback diode, and no coil.** The card has no inductive load, so there is no D1.

Both rails are switched, and for the reason §4.2 gives: +5 V alone leaves the Gotek parasitically powered through the pulled-up signal lines, and GND alone upsets the other drive. The relay-versus-silicon decision does not touch that — it decides *what* switches the rails, not *which* rails are switched.

4.5 The panel switch, and the one lead that leaves the board

The panel end is a **flip switch: UP = the physical drives, DOWN = the Gotek.** It is a plain SPST in series between the cable's +5 V and the control node, so:

Switch	Lead	`CTRL`	DS-A goes to
UP (open)	floating, R1 pulls down	**low**	**the physical drive**
DOWN (closed)	tied to +5 V	high	the Gotek

Up being the fail-safe position is not a coincidence of wiring — it is the same property the relay's de-energised contact had, and it is why R1 exists. The lead carries microamps (three CMOS inputs and a gate), where the relay's coil wanted 73 mA, and the two wires are interchangeable: reversing them changes nothing, because an SPST in series with a gate input is symmetric.

R5 (1 kΩ) is in series in that lead, and it is fitted, not deferred. The lead leaves the board and runs to the front panel, so it is the one net here that can arrive carrying something the board never sent — a transient, or a short to something else in the machine. 1 kΩ limits what that can push into three CMOS gates and a FET gate; the FET's ~630 pF gate then turns in about a microsecond, which is irrelevant for a switch a hand moves. R1 sits on the **control** side of R5, so the fail-safe is unchanged: an unplugged lead is still a low.

The card-end connector is DECIDED (2026-10-22): a bare 2-pin 2.54 mm header, with a female Dupont housing on a **twisted pair** of leads to the panel switch. The maintainer's reasoning, recorded because it is the reason: *"just a pin header is fine — an appropriate switch with a twisted pair dupont connector wire is trivial to implement."* A series SPST has no polarity to key,

the two wires are interchangeable, and a lead that works loose fails to the safe position; twisting the pair is what keeps the lead quiet next to the floppy cables.

Considered and rejected, recorded so it is not re-litigated: a shrouded **JST PH 2.00 mm** (`S2B-PH-SM4-TB`), the part the WiFi COMM PAC uses for its keyboard-adapter power port, where the shroud and polarising slot are what remove the reverse-voltage case. It would buy retention and no exposed 5 V pin, and cost PH housings and a crimp — for a two-wire series switch that does not need either.

4.6 Termination and pull-ups (why drive A matters)

The O1 terminates the floppy bus with a **150 Ω resistor pack on every signal line, located in physical drive A** — the drive at the far end of the cable. Drive B has no termination (Loxley parts 5 and 12). Two consequences:

- The shared signal lines stay terminated in both modes, because only DS-A is switched: drive A remains connected to every other line, so its 150 Ω pack stays on the bus even when the Gotek is selected.
- **DS-A loses its pull-up in Gotek mode.** The 150 Ω pull-up lives at drive A; routing DS-A to the Gotek disconnects it. The Gotek has **1 kΩ pull-ups on most lines but none on the drive-select lines** (Loxley part 5), so a floating DS-A is exactly the failure Loxley chased for days.

→ Add an on-card pull-up on the Gotek-side DS-A net (U1′s B1 → J3.10): **1 kΩ default, with a 150 Ω option** (jumper or second footprint) to match the O1′s own termination. This is the single most important addition to the design, and it exists only because the Gotek replaces drive A rather than drive B. Loxley tried both 1 kΩ and 150 Ω on the select lines.

5. Reference schematic (Rev A, netlist level)

Components:

Ref	Part	Notes
J1	2×17 socket, 2.54 mm	bottom, mates O1 mainboard header
J2	2×17 header, 2.54 mm	top, OEM cable ⟶ physical drives
J3	2×17 header, 2.54 mm	top, Gotek (signals only)
J4	4-pin header (floppy power)	switched +5 V / GND to Gotek
J5	2-pin header	front-panel switch lead — bare 2.54 mm pins for a Dupont housing; JST PH alternative in §4.5
U1	**NC7SB3157P6X**, SC70-6	DS-A 2:1 switch (§4.4)
U2	**TPS22918**, SOT-23-6	Gotek +5 V switch; `CT`/`QOD` open
Q1	**AO3400A**, SOT-23	Gotek GND switch
R1	10 kΩ	control-node pull-down (fail-safe)
R5	1 kΩ	**in series in the switch lead** (§4.5)
R3	1 kΩ	DS-A pull-up, Gotek side, default (§4.5)
R4	150 Ω (DNP)	DS-A pull-up alternative — fit R3 **or** R4 (§4.5)
R2, D2	1 kΩ + LED (optional)	front-panel "Gotek active" indicator

Connections (pin n of J1 unless stated):

- **Pass-through:** J1.n ⟶ J2.n for every pin **except 10**. This carries the signals *and* the +5 V / +12 V / GND rails to the physical drives unchanged.
- **DS-A:** J1.10 ⟶ U1 `A` (the common). U1 `B0` ⟶ J2.10 (physical A, the **S-low** throw). U1 `B1` ⟶ J3.10 (Gotek, the **S-high** throw).
- **DS-A pull-up:** R3 (1 kΩ default; 150 Ω option) from the J3.10 net to +5 V, on the Gotek side of the switch (§4.5).
- **Gotek signals:** J1 pins 8, 18, 20, 22, 24, 26, 28, 30 (and optionally 32, 34) ⟶ the matching pins of J3. J3 pins 12, 14, 16 and all GND/power pins are **NC**.
- **Gotek +5 V:** J1 +5 V (21/23/25) ⟶ U2 `VIN`; U2 `VOUT` ⟶ J4 (+5 V).

- **Gotek GND:** board GND ⟶ Q1 source; Q1 drain ⟶ J4 (GND).
- **Control:** J1 +5 V ⟶ J5.1 ⟶ (panel switch) ⟶ J5.2 ⟶ **R5** ⟶ `CTRL`, which drives U1 `S`, U2 `ON` and Q1′s gate. R1 pulls `CTRL` to GND. **UP = physical drives, DOWN = Gotek** (§4.5). Optional R2+D2 from `CTRL` to GND for the indicator.

6. BOM (Rev A)

Qty	Item	Notes
1	**NC7SB3157P6X** 2:1 analog/bus switch, SC70-6	DS-A; ≤10 Ω, break-before-make. ~$0.35–0.60
1	**TPS22918** load switch, SOT-23-6	Gotek +5 V; 52 mΩ, 2 A. $1.60 @1, $0.51 @5
1	**AO3400A** N-MOSFET, SOT-23	Gotek GND return; 28 mΩ at Vgs = 4.5 V. ~$0.10
1	2×17 socket, 2.54 mm, vertical	bottom mezzanine
2	2×17 header, 2.54 mm, vertical	top (cable + Gotek)
1	4-pin header (floppy power)	Gotek power, right-angle
1	2-pin header	panel switch
1	10 kΩ TH	control-node pull-down (R1)
1	1 kΩ TH	series resistor in the switch lead (R5)
1	1 kΩ TH	DS-A pull-up (R3, default)
1	150 Ω TH (DNP)	DS-A pull-up alternative (R4)
1	1 kΩ + LED (DNP)	optional indicator
1	PCB, 2-layer, 1.6 mm	ENIG preferred (mating, hand-soldered SMD)

7. Physical design brief

Board (PCB). - 2-layer, 1.6 mm FR4; ground plane on the bottom layer; +5 V/GND traces ≥ 0.5 mm (Gotek ~0.3–0.5 A). The Gotek's return path now runs **through Q1**, so the GND_SW trace and Q1′s pads carry that current too. - Envelope to be fixed against the mainboard keep-outs (`research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf`) and host photos (`research/pictures/`); target a compact card that clears the DD board. **Height is the binding constraint**, not area — it is what decided the switch (§4.3), and the three SMD packages are ~1.5 mm against a relay's 10 mm. - J1 (socket) on the bottom face, positioned to engage the mainboard 34-pin header; add 2 mounting holes/standoffs so the mezzanine cannot rock. - J2 and J3 on the top face; orient both so the OEM ribbon and the Gotek ribbon exit without fouling. Silk-label each connector, mark pin 1, and label "O1 IN", "⟶ DRIVES", "GOTEK", "GOTEK 5V", "SWITCH". - Assembly is **mixed**: the connectors and the four passives are through-hole, the three switches are SOT-23/

SOT-23-6/SOT-363. Order the boards with paste and a stencil or expect to hand-solder SC70 — it is the smallest part on the card.

Gotek mount (3D-printed part). - The Gotek (top cover removed) fits a floppy storage pocket; with its top headers fitted it is too tall, so connect it by a slim pigtail/direct-wire pads rather than plugging a header on top. - Design a printed bracket/faceplate that holds the Gotek in the pocket at the right depth, exposes the USB port, and carries the panel switch. This is the "Modification to support Gotek drives with 3D-printed floppy-disk faceplate" write-up planned in `site/README.md`. **Cable egress from the floppy pocket — two accepted routes.**

The card itself sits at the logic-board connector, inside the machine. What has to leave is the **Gotek run** (signals + switched power) and the **2-wire switch lead**, and the pocket's ventilation slots are the constraint. Both of these are accepted, and the maintainer's guidance is to pick one:

1. **Break the 34-pin run into parts that fit the ventilation slots** — Loxley's approach. The full-size ribbon is split into narrower runs (or pulled down to the reduced 12-line signal set of §5 plus the two power wires) so each part passes a slot, and they are rejoined inside the pocket. **Nothing on the machine is cut.** At the card end that is still a 34-way IDC socket onto J3, with only the needed conductors fitted — the unused positions simply carry no wire.
2. **Cut the case** to pass a full-size 34-pin cable, alongside the two switch wires. More work, irreversible, and it gives a clean unbroken ribbon with no joints to go intermittent.

The switch lead is two wires and passes either way; it is the ribbon that decides which route is chosen. **Both routes need the same card**, so this does not change the PCB — it changes what the builder does at the pocket, and it is why J3 is a full 2×17 header rather than a 12-pin one: the cable that arrives may be either.

Panel switch. - Plain panel-mount **SPST flip switch: UP = physical drives, DOWN = Gotek** (§4.5). 2-wire lead to J5 through **R5**; the lead carries microamps, never drive power. Mount next to the Gotek pocket so the up/down reading matches the machine's own "which drive" convention.

Reversibility. No cable or machine modification: unplug the card and the machine is stock.

8. Decisions, and what each one rests on

All eight are closed, and two of them — item 3 (the switch) and item 8 (the switch connector) — were settled by the maintainer rather than by a session. Each carries the evidence it rests on; the two that remain bench-verifiable are flagged as such.

1. **Drive topology — CONFIRMED (user, 2026-10-10).** The switch selects **drive A** (physical A ↔ Gotek); **drive B stays present** in both modes.
2. **Connector genders — CONFIRMED.** The O1 logic board's floppy connector is **P8, a 34-pin MALE header**. The Field Service Manual (2F00040) instructs "Attach the disk harness to the 34-pin connection on the logic board … Be careful not to bend any pins", and the double-density adaptor's "sockets … fit onto the logic board disk harness connector (P8)". The harness is a **female IDC socket** and is **not keyed** — "the RED stripe on the harness must be to the

RIGHT" fixes pin 1. Therefore **J1** = **female 2x17 socket** (card bottom) and **J2** = **male 2x17 header** (OEM cable). Pin order is confirmed in §2.

3. **Switch element — SETTLED BY THE MAINTAINER, 2026-10-22: solid state.** The decision, and the constraint that decides it:

   There is not enough height in the case for relay blocks.

   ~10 mm for a DIP-14 signal relay, 35 mm for a socketed industrial one, on a card that is a mezzanine between the logic board and the floppy harness. So: **three silicon parts** (§4.4) — **NC7SB3157P6X** for DS-A, **TPS22918** for the Gotek's +5 V, **AO3400A** for its ground return — ~$3, no coil, no click, no height. The fail-safe survives: the switch's own truth table is *S low* → *B0*, and B0 is the physical drive. The distribution check agrees: **Omron G6A-434P** / **G6A-474P** were discontinued in 2010, and the whole 4-pole DIP class with it (Panasonic TQ4-5V and DS4E-S-DC5V are broker-only at $20–190).

4. **Isolation depth — DECIDED: Loxley's proven minimum.** Switch **DS-A + both power rails** and nothing else. The shared signal lines are *not* additionally gated: doing so adds series elements to a 4 MHz-class bus for no proven benefit, and the failure Loxley actually chased (a Gotek that would not power down) is fixed by the dual-rail switch alone. Full signal gating is recorded as a Rev B experiment, not Rev A scope.

5. **DS-A pull-up — CONFIRMED 1 kΩ default, 150 Ω option.** The FSM settles the termination location: "The A drive has an 8 pin 150 OHM Terminator resistor pack. B DRIVE DOES NOT." So when DS-A is routed to the Gotek it loses the O1′s 150 Ω pull-up, and the Gotek has no pull-up on its drive-select lines (Loxley part 5). The card therefore fits **R3 = 1 kΩ** as the default and **R4 = 150 Ω** as an alternative footprint (fit exactly one) on the Gotek-side DS-A net.

6. **Gotek drive-select jumper — SETTLED (bench-verify).** The Gotek must answer the O1′s **drive-A** select, which is cable **pin 10**. The Gotek manual labels the jumpers `S0 = Drive Select 0` / `S1 = Drive Select 1`, and FlashFloppy's generic guidance for a Shugart host is "unit `0` → place the selection jumper at **S0**". The O1, however, straps its drive A as **DS1** (FSM 8.1.4.11: "Drive A has a jumper at positions HM and DS1"), and the proven O1 adapter (`WayneVisser/Osborne1FloppyAdapter`) notes "Jumper only **S1** on the GoTek". **Start with S1**; if the Gotek is not seen, move to S0. This is a 30-second bench check and the card is agnostic to it.

7. **Gotek cable route / bracket — DECIDED.** Gotek in the **right-hand floppy storage pocket** (Loxley part 14), driven from a **reduced ~12-line pigtail** that passes the pocket vent slots (a full 34-pin ribbon does not fit), with a 3D-printed faceplate/bracket carrying the Gotek and the panel switch. This is the "Modification to support Gotek drives with 3D-printed floppy-disk faceplate" write-up planned in `site/README.md` (§7). The two accepted egress routes — break the 34-pin run into parts that pass the slots, or cut the case for a full-size cable — are in §7, and the card is the same either way.

8. **Panel-switch connector — DECIDED BY THE MAINTAINER, 2026-10-22: a bare 2-pin 2.54 mm pin header.** *"just a pin header is fine — an appropriate switch with a twisted pair*

dupont connector wire is trivial to implement." The lead is a series SPST (§4.5), so there is no polarity to key and the two wires are interchangeable; the shrouded JST PH alternative is recorded in §4.5 as considered-and-rejected rather than left open.

The evidence §8 rests on

- **O1 Field Service Manual 2F00040** (`research/osborne1/`): 150 Ω terminator is on drive A only; logic-board connector P8 is male; harness not keyed, red stripe = pin 1; drive A strapped DS1.
- **O1 mainboard schematic, DWG 1A2011-00 Rev E sheet 9 of 9 (DISC CONTROLLER)** (`research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf`, PDF page 12): the P8 34-pin header — GND = `1,2,3,5,9,19,27,29,31,33`; +12 V = 11,13,15,17; +5 V = 21,23,25; **pin 4 = TG43 and pin 6 = EARLY, neither is GND**; pins 7 and 14 are not drawn at all; DS-A/DS-B on 10/12. Read at 600–1200 dpi; see `docs/o1-mainboard-schematic.md` §5 (#62).
- **O1 disk-electronics schematic, DWG 1A3004-00/01 Rev C sheets 1–3** (PDF pages 1–3 of the same file — a *different* drawing, the drive's own board): head/drive connectors P1/P2; P2 lists GND = `1,3,5,7,9,19,27,29,31,33` and CASE = 32. It disagrees with P8 on pin 2 versus pin 7, which is harmless — neither is a driven signal on the O1, so the card ties both to GND.
- **Gotek SFR1M44-U100K user manual**: jumper table `S0 = Drive Select 0`, `S1 = Drive Select 1`, `MO = Motor On`, `JA = READY on 34-pin interface`.
- **FlashFloppy wiki — Initial Setup / Gotek Models**: Shugart hosts expect unit 0 ⟶ jumper `S0`.
- **Omron G6A-434P** datasheet: 4PDT DIP-14 through-hole relay; 5 VDC coil = 72.5 mA / 69 Ω / ~360 mW; contacts 1 A 30 VDC (ST-US). The part is **obsolete** — G6A-434P and G6A-474P were discontinued with a last-buy date of 2010-06-29, and the 4-pole DIP class went with them (§4.3).
- **Maintainer, 2026-10-22 — the constraint that decides the switch:** *"There is not enough height in the case for relay blocks."* Recorded verbatim because it is the load-bearing fact: the card is a mezzanine, and a DIP-14 relay is ~10 mm and a socketed industrial 4PDT is 35 mm. It settles §4.3 without reference to price or availability.
- **NC7SB3157 / FSA3157 datasheet** (Fairchild, Rev 1.0.4 — read as text, not as a PDF figure): function table **"Logic Level Low ⟶ B0 connected to A; Logic Level High ⟶ B1 connected to A"**; pin assignments in SC70-6 **1 = B1, 2 = GND, 3 = B0, 4 = A, 5 = VCC, 6 = S**; 1.65–5.5 V VCC, <10 Ω typical on-resistance, rail-to-rail signal handling, control input tolerant to 7 V, and **break-before-make** select circuitry. The truth table is what makes B0 the fail-safe throw, and the break-before-make is what keeps the two devices from ever sharing DS-A.
- **TPS22918 datasheet and product page** (TI SLVSD76C): 1–5.5 V in, 2 A, 52 mΩ at 5 V, **N-channel switch with a charge pump**, adjustable rise time on `CT`, and a configurable quick-output-discharge `QOD`. The DBV-6 pinout used here (`1 VIN, 2 GND, 3 ON, 4 CT, 5 QOD, 6 VOUT`) is the one KiCad's own library carries for **TPS22917DBV**, the same package and family.
- **Distribution check, 2026-10-22** (DigiKey / Newark / LCSC / TME via their product pages and the Findchips and Octopart aggregators): G6A-434P-ST-US and G6A-474P-ST-US obsolete;

TQ4-5V broker-only $20–31 CAD; DS4E-S-DC5V broker-only $21–186. In production and stocked: **AQY212GS** (Newark $3.05 @1, $2.22 @100); **TPS22918** ($1.60 @1, $0.51 @5, $0.15 @6k); **SN74CBT3257** ($1.13 @1); and, for the relay routes that were considered and rejected, **G6S-2 DC5** (Newark $6.00 @1, $4.65 @100), **HFD2/005-S-L2** (Newark $2.20 @1, $1.66 @100), **FTR-B4CA005Z** (Electro Sonic $2.37 @50), **MY4N-DC12-S** (TME $7.78 @12) and Finder 55.34 (TME $13.59 @1).

- **Richard Loxley, Osborne Restoration parts 5 and 12–15**: 150 Ω pack at the far drive; Gotek has 1 kΩ on most lines but not on drive select; switching only +5 V leaves the Gotek parasitically powered, only GND upsets the other drive — switch both.

9. Rev A schematic capture

Captured 2026-10-10 as `hardware/floppy-adapter/floppy-adapter.kicad_sch`, generated by `gen_floppy_adapter_sch.py`. **Rewired, footprinted and re-verified 2026-10-22** — the same treatment the WiFi COMM PAC got (#347) — and then **re-captured the same day for the solid-state switch** (§4.3, §4.4).

- **J1** female 2x17 socket (O1 logic-board P8), **J2** 2x17 header (OEM cable), **J3** 2x17 header (Gotek signals), **J4** 1x04 header (switched Gotek power), **J5** 1x02 header (panel switch), **U1** NC7SB3157P6X (DS-A), **U2** TPS22918 (+5 V), **Q1** AO3400A (GND), **R1** control pull-down, **R3/R4** DS-A pull-up, **R2+D2** optional indicator (R2, D2 and R4 are `dnp` — on the board, not in the BOM).
- Every J1 pin except DS-A (10) is **drawn straight through to J2**. DS-A goes to U1′s common; its **B0** throw returns to J2.10 and its **B1** throw goes to J3.10. `+5V_SW` and `GND_SW` reach the Gotek only via U2/Q1 ⟶ J4; the J3 header is signals-only, and +12 V, 4 MHz and pin 14 never reach the Gotek.

The capture's guarantees, and the hazards it removes

the hazard	the capture
a net **label on every pin**, and no wire between any two parts	**every net drawn** as orthogonal segments on a 1.27 mm grid, routed by the Manhattan A* in `gen_floppy_adapter_sch.py`
103 labels doing the connecting	**one label per signal net** (21), so the names survive and a missing wire *breaks* the netlist instead of being hidden
`GND`/`+5V`/`+12V` as labels	routed like signals, with **one power symbol per rail** — a symbol per pin would stack 3.81 mm glyphs 2.54 mm apart on J1's eleven ground pins — plus a **PWR_FLAG** per rail
every `Footprint` field empty	**13 of 13 assigned**, from KiCad's own libraries
a 4PDT relay, `K1`, with no footprint and logical pin numbers	**three silicon switches** whose pin numbers are the parts' own (§4.3, §4.4)
nothing regenerated or compared	`tools/check_schematics.py` now asserts the **pinout of J1 and J2**, **which throw of U1 is which device**, the control node's three consumers, the connector and switch footprints, and that the committed file matches its generator

Verified, not asserted:

- `kicad-cli sch export netlist` matches §5's net table; `kicad-cli sch erc --severity-all` is **0 errors / 19 violations**, all 19 the one kind (embedded `local:` symbols, one per placed symbol).
- `kicad-cli sch upgrade` — the operation that detached 14 pins on the WiFi board — leaves the netlist **identical**. 0 overlapping and 0 duplicate collinear segments, because the generator normalises its own geometry.
- `python3 tools/check_schematics.py --board floppy-adapter` and `python3 tools/check_boards.py` pass.

Three defects came out of the capture, all found by comparing netlists rather than by looking at the sheet:

1. **A wire ended on a no-connect pin.** The router blocked other nets' pins but not the pins marked no-connect (they carry no net, so they are not in the net-holding table), and the first wired netlist grew `J3.14` and `K1.11` onto `FLP_DS_A_GOTEK`. A no-connect marker says the pin goes *nowhere*; the netlist believes the wire. The sibling generator in `hardware/wifi-comm-pac/` had the same latent hazard and was given the same fix.

2. **`tools/check_schematics.py` was carrying a pinout table that contradicted the design** — it read "pins 2/4/6 are GND" against §2′s "pins 4 and 6 are TG43 and EARLY, never ground". Nothing called the table, so nothing noticed; it is now the check for J1 and J2, which is what makes it true or false.
3. **The relay's pin numbers were placeholders and its footprint was missing**, which is what the switch decision above finally closed — with the parts' own pin numbers, taken from their datasheets and cross-checked against KiCad's library symbols.

State: the schematic is complete — the next step is the PCB

Every decision this design needed is made (§8), every net is drawn, every part has a footprint, and the two things that were left hanging — the switch part and the switch connector — are closed. **The board is ready for PCB floorplan and routing**, which is the next unit of work: place the connectors on their faces, set the keep-outs against the mainboard and the pocket, and route it.

The one thing the floorplan must respect is the reason the switch is silicon: **height is the binding constraint**, and the three switch packages are ~1.5 mm against a relay's 10 mm.

10. References

- Issue #62 (this design); feeds #20 (floppy interposer). Disk formats: `docs/dd-upgrade-formats.md` (#95).
- `research/osborne1/floppy-adapter/` — Wayne Visser passive adapter + PROVENANCE.
- `research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf` — mainboard schematic.
- `hardware/floppy-adapter/README.md` — concept summary.
- Richard Loxley, Osborne Restoration **part 5** (150 Ω term pack at the far drive; Gotek has 1 kΩ on most lines but *not* drive select) and **parts 12–15** (A drive has 150 Ω, B has none; DPDT + relay switching lesson).

Serial and the modem

Status: Solved. Both connectors and their signals are read from the Technical Manual's own §4.2/§4.3 tables and confirmed against schematic sheet 7, which carries the DE-9P pin numbers through to the on-board P1 header unchanged. Two things are still open, both named in place below: the **physical orientation** of the 2×5 P1 header on the board, and which part the **RXD receiver** is.

P1 is the MODEM port — a DE-9P at TTL levels, not RS-232, which is what makes it the clean attachment point for a modern serial device: `TXD` and `MCB` are 5 V TTL, `RXD` is a bipolar input (negative for mark), `MSB` is an open-collector modem-status bit, `CTS` is the only hardware flow-control line, `RI` is the ring indicator, and **pin 7 is a real +12 V supply** — the Astec rail is rated `2.02 A`, and pin 7 reaches it **through `R21` (22 Ω)**, a resistor the keyboard branch shares. **P2 is the RS-232 port — a DB-25S wired as RS-232C-compatible DTE**, with `CB`, `CC` and `CF` held at +5 V because the machine does not use them. Both connectors are the same **6850 ACIA**, at status `2A00H` / data `2A01H`.

The rest of the document is the pinout evidence (§4.2, §4.3 and the schematic-confirmed P1 header), the design notes the WiFi modem (#56) has to satisfy, the baud-rate options the archive documents (#21), and the 6850′s framing byte in the BIOS.

Source: Osborne 1 Technical Manual, ch. 4 "Osborne 1 Interface Design" — §4.2 SERIAL RS232 INTERFACE, §4.3 MODEM — archived in-repo at `research/osborne1/modem/o1techman-interface-extract.pdf` (printed pages 13–22), with schematic sheet 7 of `research/osborne1/OCC1_1A2011-00_Schem_RevE.pdf` for the on-board header.

This is the authoritative connector/pinout reference for the WiFi-modem work (#21, extended by #56) and for the severable modem module (#56 §E / COMM PAC variant). The memory-map and ACIA register detail live in `docs/o1-memory-io-map.md` §"6850 ACIA detail"; this document adds only the **physical connector** information that doc does not carry.

4.2 RS-232 serial interface (P2)

- Configured as **RS-232C-compatible DTE**, but *"certain of the RS-232C signals are held at +5 volts since they are not needed to control the Osborne 1."* A **6850 ACIA** drives the port.
- Port addresses: **status `2A00H`, data `2A01H`** (shadow mode). Status bits (Fig 4.2): `RDRF`, `TDRE`, `DCD`, `CTS`, `FE`, `OVRN`, `PE`, `IRQ` — see the 6850 datasheet for full semantics.

DB-25S pinout (Fig 4.2.2)

DB-25	RS-232	Definition
1	AA	Frame ground (optional)
2	BA	Transmitted data (low = 1)
3	BB	Received data (low = 1)
4	CA	Request to send (high or no connection enables)
5	CB	Clear to send (**always high on OCC 1**)
6	CC	Data set ready (**always high on OCC 1**)
7	AB	Signal ground
8	CF	Received line signal detected (**always high**)
20	CD	Data terminal ready (high or no connection enables)

Pins 9–19, 21–25: no connection. Only the listed pins are wired.

4.3 MODEM interface (P1) — TTL levels, not RS-232

> "The modem and RS-232 interfaces are basically one and the same. In addition to the serial port, **TTL-level signals may be directly input into the 6850 ACIA using the modem port connection.**"

The MODEM jack is the clean way to attach a TTL-serial device (an ESP32 WiFi modem, our module) **straight onto the ACIA** without an RS-232 level shifter. Read/write it via the CP/M `IOBYTE` function, same as the serial port.

4.3.1 Signal direction

Osborne 1	dir	Modem device
GND	—	GND
TXDATA	→	TXDATA
RXDATA	←	RXDATA
MSB	←	MSB
CTS	←	CTS
MCB	→	MCB
+12	→	POWER IN
RI	←	CD

4.3.2 DE-9P pinout (“standard numbering of the DE-9P connector”)

Pin	Signal	Definition
1	GND	Signal ground
2	TXD	Transmitted data — **TTL logic, 1 = high**
3	—	Not used
4	MSB	Modem status bit — open collector, 50 µA sink = inactive
5	CTS	Clear to send
6	RXD	Receive data — **bipolar input, −0.5 V…−10 V = 1**
7	+12 V	**+12.6 V DC** from the `2.02 A` rail, **through `R21` (22 Ω)**; the Comm-Pac manual labels it `VDD`
8	MCB	Modem control bit — TTL, **low suppresses output**
9	RI	Ring indicator — TTL, high-to-low sets flag

Design notes for #56

- **Levels are asymmetric, and the receive direction is not free.** TXD and MCB are **5 V TTL** because the 6850 and the PIA are +5 V parts; the ESP32-S2's inputs are **not 5 V tolerant** (`VDD + 0.3`), so both need a translator. CTS, RI and MSB leave the module at TTL thresholds; MSB is open-collector and needs an **open-drain** driver. **RXD is a bipolar input** (−0.5…−10 V = mark): on sheet 7 the RXDATA line enters a receiver with zener clamp **CR1 (1N5231B, 5.1 V)** — a comparator-style front-end, not a TTL gate. So driving RXD with 0–3.3 V leaves the mark at or below threshold, and the module must swing **negative** for a clean mark. A MAX3232 on 3.3 V (its own charge pump, ±5.5 V) is the cheap answer, and the negative swing still wants confirming on real hardware — see the open item below. The receiver's part number is **open**: `docs/o1-mainboard-schematic.md` sheet 10 lists **UE3 as a 1489** with three **LM3900** comparators at UD4, while a pin-level read of the same region returned **UE3 as an LM1458**. It does not change the requirement (negative for mark), but it changes the input impedance, so re-read that region at native resolution before fixing the drive current.
- **P1 pin 7 is the `+12V` net through `R21` (22 Ω), and that resistor is the limit on this port.** The Technical Manual's §4.3.2 table says it in words — "Connected to power supply through 22 ohms" — and sheet 7 shows the arrangement: `R21` hangs off the `+12V` rail and its lower end is a junction that feeds **both** this port's pin 7 and **`J6 → P4-19`**, the keyboard branch. The two loads therefore **share** one 22 Ω resistor.

  A 22 Ω feed is a hard current limit, so this is a low-current supply. At 500 mA it would drop 11 V, which no 12 V rail survives; at 100 mA the drop is 2.2 V and about 10 V reaches the load. A period Bell-103 modem with its own DAA draws tens of mA, which is what makes a 22 Ω feed possible at all — and the Comm-Pac's own manual quotes **no** supply current anywhere (its specification page gives the modulation, the carrier frequencies and the line levels, and nothing about current). So the port's budget is the one #56 already assumes: small, and shared with the keyboard adapter. The eleven-lamp panel is sized at 2–3 mA per lamp for exactly this reason.
- **MSB (pin 4) is open-collector** and **MCB (pin 8) low suppresses output** — together these are the modem's "hook"/enable handshake. RI (pin 9) falling edge sets the ring flag; the ESP32 can drive it to signal an incoming "call" (AT RING) to the O1.
- **CTS is the only hardware flow-control line** brought out (pin 5).

On-board P1 header — schematic-confirmed

Schematic sheet 7 (SERIAL I/O, `OCC1_1A2011-00_Schem_RevE.pdf`) draws the P1 (MODEM) connector on the right edge with the **DE-9P pin numbers carried through unchanged** — there is no renumbering between the case DE-9 and the on-board P1 header:

P1 pin	Net	P1 pin	Net
1	GND	2	TXDATA
4	MSB	5	CTS
7	+12V via R21 22Ω — see *Design notes for #56*	6	RXDATA
8	MCB	9	RI
3	(NC)		

Physically P1 is a **2×5 IDC male header** on the board, cabled to the case DE-9P with a straight ribbon (1:1, no conductor swap). A straight IDC crimp interleaves the DE-9 rows, so on the 2×5 the signals land as: row A (1-2-3-4-5) = DE-9 **1,3,5,7,9** = GND, NC, CTS, +12V, RI and row B (6-7-8-9-10) = DE-9 **2,4,6,8,—** = TXDATA, MSB, RXDATA, MCB, NC.

> The schematic confirms the **signal→pin** mapping (the table above). The **physical orientation** of the 2×5 header on the board (which end is pin 1) is a layout detail not in the schematic — confirm against the PCB silkscreen / a real board before committing a PCB footprint. Same class of in-hardware validation as #47.

Also on sheet 7 (for the record): the ACIA baud clock comes from **UC3 (LS161)** dividing a `4USEC/2USEC` tap (jumper **J1** selects the rate), and **MODEM RI** routes to the video-PIA **UC15 CA2** input (sheet 5) — consistent with `docs/o1-memory-io-map.md`.

Baud-rate options the archive documents (#21)

The stock machine gives 300/1200 (J1 in) or 600/2400 (J1 out), and the factory documents 2400/9600 as a modification — the jumper and divider detail is in `docs/o1-memory-io-map.md` §"6850 ACIA detail". Two further things come from *The Portable Companion* #8 (July 1983).

The factory modification, as the Companion cites it

The issue's "**The CRT Connection**" box (PDF p. 87, printed 85) tells a user who wants an 80-column terminal to *"set your computer for 9600 baud using Osbaud or the method shown on page 22 in the Technical Manual."* That page is the manual's **Figure 4.4 "Baud Rate Hardware Modification"** — the LS161/LS163 at C3, pin 2 moved, J2 fitted, J1 *"DON'T CARE"* — quoted in `docs/o1-memory-io-map.md` §"6850 ACIA detail". So the Companion's pre-SCREEN-PAC answer to "how do I get 80 columns?" was the **serial port plus the factory's own baud mod**, not a video board.

The CRT recipe itself

1. Set the machine for **9600 baud** (as above).
2. Connect a standard serial cable between the terminal and the computer. The cable is only **pins 1, 2, 3 and 7, straight through** (1–1, 2–2, 3–3, 7–7).
3. Set the terminal's DIP switches to **1 start bit, 1 stop bit, 8 data bits, no parity, full duplex**.
4. `STAT CON: = CRT:` to make the terminal the console.

5. *"The end result is a 80-column, 9600 baud refreshed, block cursor screen."* The procedure is *"untested with terminals other than Televideo"* — the reviewer's was a **Televideo 920C**.

Advent Products "Osbaud" — a clock substitution, not a bus interposer

Reviewed by **Brad Baldwin** in the same issue (PDF p. 87, printed 85; extract at `research/technical-articles/TPC-Issue08-1983-pp87-osbaud-baud-rate-generator.pdf`). $59.95, **16 DIP-switch-selectable baud rates**, used by the reviewer at **110** (an ASR33 Teletype), **9600** (the Televideo 920C) and **19200** (a $10,000 high-speed band printer), with mainframe up/downloads at high speed. It is *"a small auxiliary circuit board … soldered to the main logic board **in the area behind the serial port**"*, with the DIP switches reached through **a small slot cut in the venting grill** and a self-adhesive rate chart stuck to the keyboard. Advent even supplies the .050" Allen wrench for the brightness and control knobs, and the package covers both case colours (blue/gray and black/brown). Installation wants a Phillips screwdriver, an **Xacto knife to cut across a circuit trace**, and a low-wattage soldering iron — and, like any such product, **voids the warranty**.

The four solder points, decoded

*"Osbaud is soldered to **pins 4 and 12 of the 6850 UART**, **pin 7 of the 74LS32**, and **pin 7 of the 74LS04**."* Read against the schematic (`docs/o1-mainboard-schematic.md` sheet 10, and §"6850 ACIA detail" in the memory map):

Solder point	Is	Why
6850 **pin 4**	**TxCLK** — strapped on the board to pin 3 (RxCLK)	the clock *input*: one wire drives both halves of the ACIA
6850 **pin 12**	**Vcc**	+5 V, taken from the ACIA's own supply pin
74LS32 **pin 7**	GND	ground
74LS04 **pin 7**	GND	ground — and this one is **UB3**, the inverter inside the ACIA's own baud divider

So OSBAUD does not tap the bus, needs no address, and does not use the LS161′s socket: it **replaces the ACIA's clock**, powered and grounded from the ACIA's immediate neighbourhood. That makes the Xacto knife unambiguous — the trace to cut is the one carrying the factory clock (UC3′s `QD`) to the ACIA's clock pins, so that OSBAUD's oscillator is the only driver on that net.

What the design has to be

The review gives the connections and the rate count, not a schematic, so the rest is inference — but tightly constrained:

- The ACIA still divides by 16 or 64 internally (the factory's Figure 4.4 is captioned *"SPEED SELECTED W/IN 6850"*), so OSBAUD's clock is **16× the marked rate** on the ÷16 setting — 800 Hz … 307.2 kHz across 50 … 19200 baud.

- Sixteen rates off a DIP code means sixteen selectable taps, but **a single binary chain from one crystal cannot produce the odd rates** of the standard 16-rate list (110, 134.5, 2000), so the divider has to be **preset/programmable** — the same trick the factory divider next door uses, its LS161 being a preset-load counter (`RCO` ⟶ inverter ⟶ `/LD`).
- The three rates the reviewer names (110, 9600, 19200) all sit inside the classic 50/75/110/134.5/150/300/600/1200/1800/2000/2400/3600/4800/7200/9600/ 19200 set — the same list Nuevo's optional generator advertises ("16 baud rates from 50 to 19.2K", `docs/o1-upgrade-boards.md`).

Open: the crystal and divider topology, whether the ACIA is left in ÷16 mode, and where the **74LS32** sits on the board (the LS04 is UB3 on the serial sheet; a 74LS32 near the serial port has not been located on the schematic yet). The OSBAUD instruction sheet — which the review calls thorough — or a photo of a fitted board would settle all three; worth watching for on the second-hand market.

Framing — the 6850 control byte in the BIOS (#21, #56)

Baud rate is only half of what a modem personality has to present. The other half is the **frame** — data bits, stop bits and parity — and OCC documented how to change it. "**Programming the serial port — modifying the serial port for scientific test devices**", signed by the **Osborne Computer Corporation System Technical Support Group**, in *The Portable Companion* #7 (June 1983, PDF pp. 56–57 = printed 54–55; extract `research/technical-articles/TPC-Issue07-1983-pp56-57-programming-the-serial-port.pdf`) is a worked BIOS patch:

- The factory default is **8-bit data, 1 stop bit, no parity**.
- The change is a **single control byte written into the BIOS image**: build a relocatable CP/M (`MOVCPM 60 *` for a **1.3** BIOS, `MOVCPM 59 *` for **1.4**), `SAVE 39 CPM60.COM`, `DDT CPM60.COM`, then **`S1FE9`** and put the code in at **`1FE9H`/`1FEAH`**, and `SYSGEN` it to the target disk. So the ACIA's **control-register init byte lives at BIOS offset `1FE9H`** — a second, independent handle on the ACIA beyond the `2A00H`/`2A01H` port pair, and the address to watch for #21/#56.
- **Figure 1** is a "**Table of 6850 control codes**" covering all 16 combinations at 1200 and 300 baud — e.g. **8-N-1 @1200 = `55H`**, **8-N-1 @300 = `56H`**, 7-E-1 @1200 = `49H`, 8-N-2 @1200 = `51H`; arrows mark the factory default. Read as the 6850's control register (word-select bits 2–4, clock-divide bits 0–1) the values are exactly right, which is a useful cross-check on the `docs/o1-memory-io-map.md` §"6850 ACIA detail" claim that the ACIA divides by 16 or 64 — the `55H`/`56H` default sits in the **÷16** field.
- **Figure 2** is a **which-BIOS-do-I-have** test: `DDT`, `DA900`, and look at the third byte after the `490` — `C3 xx E5` is a **1.3** BIOS, `C3 xx E1` a **1.4**.

The article does not touch the baud divider, so it is complementary to the Hogan/Figure 4.4 rate modifications above: **rate from the divider, frame from the BIOS control byte**. For the module (#56) both matter — a WiFi modem that only ever presents 300/1200 8-N-1 will fail on a machine whose owner has patched the frame for a lab instrument, which is exactly the case the article exists for.

The IEEE-488 port

Status: Solved. The P3 pinout is transcribed from the Technical Manual's §4.1.2 and corroborated line for line by the ROM's own comment block; what the port implements is read from the ROM's eight drivers and the manual's own figure. The one thing still open on P3 is electrical rather than logical — the loading a device may present (§7).

P3 is a 26-pin card edge connector carrying 24 IEEE-488 lines (pins 25 and 26 are unused), driven by a **Motorola MC6821 PIA at `2900–2903`** (§6). The machine can act as Talker, Listener or Controller, and its own bus address is **0** (§1). It implements **a subset of IEEE-488, with no three-wire handshake** — the ROM's **eight low-level drivers** are the whole of its behaviour, and `DAV`/`NRFD`/`NDAC` cannot be read back, which is why every device on this port invents its own clock (§2, §6.1). The same pins are re-labelled as a **Centronics printer port** when SETUP selects the printer personality (§6.2). §5 is the project's one fully-worked external device on the port: a **game joystick on `D0–D4`**, read either through driver 8 (`CALL 0E154H`) or by bit-banging the PIA directly.

Source: *The Portable Companion* #8, July 1983 — "**Understanding the IEEE-488 port** — Scientific interfacing with the Osborne 1" by **Wayne Kogachi** and **Shahriar Kianersi**, PDF pp. 38–41 (printed 36–39), and "**Interfacing External Devices** — add a game joystick to the Osborne 1" by **Kelly Smith**, PDF pp. 44–49 (printed 42–47). Archived in-repo at `research/technical-articles/TPC-Issue08-1983-pp38-41-understanding-the-ieee-488-port.pdf` and `research/technical-articles/TPC-Issue08-1983-pp44-49-interfacing-external-devices-joystick.pdf`.

Both authors of the first article are **OCC staff** — Kogachi a software engineer, Kianersi a system support specialist — so the article is effectively the vendor's own account of the port, and it points at the *User's Guide* chapter "IEEE-488 implementation" for the same material.

The port's address map and the ROM's view of it live in `docs/o1-memory-io-map.md` §2 (`2900–2903`, **MC6821 at UC7**) and `docs/o1-rom-source.md`; the Drive C: RAM disk that hangs off it is in `docs/drive-c-protocol.md`. **This doc records what the two articles add**: the port's device-level behaviour, the eight ROM drivers as the articles name them, and the one fully-worked application — a game joystick wired straight onto the data lines.

Bearing issues: #22 (Drive C / IEEE-488 storage), #84 (standalone Drive C on P3), #19 (virtual MX-80 / Centronics capture), #12 (interposer spec).

1. What the port is

- **TTL logic level**, driven by a **Motorola 6821 PIA**; "after configuring the pin connection for the IEEE standard, one can simply connect the Osborne to any IEEE compatible device."
- The machine can act as any of the three IEEE-488 device definitions: **Talker, Listener and Controller**. Bus management is: announce the **Talker first, then the Listener**, after which the Controller releases the bus and lets those two devices talk. The Controller may announce itself as either Talker or Listener, and its periodic takeovers "should be done *synchronously* rather than *asynchronously*."

- **Up to 15 devices** on the bus: *"The IEEE-488 port can call up to 15 different devices at one time on the data bus. (From a software point of view, the figure is 29 different devices; however, due to practical hardware limitations only 15 devices are allowed.)"*
- **The Osborne's own address is zero**, *"so no other device with that address port can sit on the data bus"* — a hard constraint on any bus we build.
- The second article's framing is the useful one for this project: *"The IEEE-488 Bus implementation is really just a **parallel data port controller** that can be used for a variety of interfacing possibilities that require byte-wide (8 bits at a time) data. This controller is known as a 6821 PIA ... and may be programmed in a variety of different input/output control modes."*

2. The eight ROM drivers

> "Osborne's IEEE-488 port has **eight low-level 'drivers,' which reside in the ROM** and take care of all the IEEE-488 interfacing requirements. These routines are documented in the *User's Guide* in the IEEE-488 implementation chapter. One may invoke any of the routines simply by calling them and passing the proper parameters through preassigned registers. One may also receive messages through assigned registers."

#	Driver	Parameter registers	Returns
1	**Control Out**	`C` = code to clear the interface (IFC) and/or set/reset REN	—
2	**Status In**	—	`A` = value acknowledging that a device is requesting service
3	**Go To Stand By**	—	lets other devices on the bus take control
4	**Take Control**	`C` = code; synchronous or asynchronous takeover	`A` = error messages, if any
5	**Output Interface Messages**	`C` = message byte (talker or listener address)	`A` = error messages
6	**Output Device Messages**	`C` = data byte; `B` = end-of-message acknowledgement code	`A` = error messages
7	**Input Device Messages**	—	`A` = data byte; `L` = error message
8	**Input Parallel Poll**	—	`A` = the parallel-poll bit pattern

Driver 8 carries a hardware consequence worth recording: *"The devices that are capable of 'parallel poll' will respond by pulling low a preassigned data line so that some kind of code is returned which will correspond to a particular device.* ***If this routine is used, only eight devices can sit on the data bus*** *(one data bit line for each device)"* — parallel poll trades device count for polling speed, 15 ⟶ 8.

The same eight, as a jump table in the ROM

`docs/o1-rom-variants.md` records that OZROM 1E removed the IEEE-488 support and that the machine-readable evidence is a jump table: *"013F-0156 WERE IEEE-488 ROUTINES".* **`013F–0156` is 24 bytes — eight 3-byte `JP` instructions, one per driver above**, which independently confirms both the count and the order: the *n*-th entry sits at `013F + 3n`, so the **eighth** (Input Parallel Poll) is at **`0154`** — and the joystick article calls exactly that address, as `0E154H`, the same routine seen in the ROM's high mapping (see §6). The articles name the drivers; the ROM fixes their addresses and their order.

Checked in the bytes, in every stock image the project holds — all of them carry eight `C3` (`JP`) triples from `013F`, the eighth at `0154`:

ROM	the first seven entries	8th entry at `0154`
factory 1.40 (`tools/emulator-setup/roms/rev1.40.ud11`)	`C3 36 09`, `C3 79 09`, `C3 8C 09`, `C3 98 09`, `C3 D8 09`, `C3 FB 09`, `C3 3D 0A`	`C3 99 0A`
factory 1.44 (`research/roms/os1-144.rom`)	`C3 49 09`, `C3 8C 09`, `C3 9F 09`, `C3 AB 09`, `C3 EB 09`, `C3 0E 0A`, `C3 50 0A`	`C3 AC 0A`
Nuevo 1.51 (`tools/emulator-setup/roms/monrom-rev1.51-12.ud11`)	`C3 3D 09`, `C3 80 09`, `C3 93 09`, `C3 9F 09`, `C3 DF 09`, `C3 02 0A`, `C3 44 0A`	`C3 A0 0A`

The parallel-poll routine moves between ROM revisions — `0AAC` in 1.44, `0999` in 1.40, `0AA0` in Nuevo's 1.51 — which is exactly why the joystick article's source must be assembled at a different origin per monitor version and then hand-patched. Same table, different targets: a usable ROM-variant signature, and the thing OZROM 1E destroys.

3. Errors the drivers report

- **Time Out** — non-fatal: *"a 100 microsecond grace period of time for the device to respond. Alternately, you can call the device repetitively to allow for slow devices to respond."*
- **No Device Present** — *"either the port is not hooked up to any device, or all the devices sitting on the data bus are turned off."*
- **Not taking the data after agreeing to take the data.**

4. A printer or plotter needs none of this

> "if only a printer/plotter is being used on the port there is no need to go through these routines. Using the SETUP program, make sure that the right protocol is chosen (Printers protocols, IEEE-488 compatible) and that the IEEE address port matches the device number. The latter can be simply accomplished with **BIOS 1.4**. With **BIOS 1.3 and earlier** versions there is a patch that is provided in the Technical Tips section of this issue. Double density computers (1.4 BIOS) have a **menu-driven device address patcher**."

That is the emulation surface for #19 (virtual MX-80): the printer path is a SETUP-selected protocol plus a device address, not a driver call sequence.

Two operational details from the same article, both worth keeping:

- **The control codes must be in capital letters.** *"Some manufacturers enforce that; otherwise an error message will be returned."*
- The MBASIC example works by poking a small assembly-language interface plus a register-save area high in RAM and calling it from BASIC (`CALL IEEE0(…)` / `IEEE1(…)`), passing the driver number and its parameters through the shared locations. That is why the BIOS-1.3-or-

earlier caveat exists — the poked routine and the ROM drivers have to coexist with CP/M in high memory.

5. The worked application: a game joystick on the data lines

The joystick article is the project's only fully-worked external device on this port, and it is small enough to reimplement: **a switch-closure box read as five data bits.**

The cable

- One **DB-9 male** (it mates an Atari joystick), one **26-pin (13-pin dual row) card-edge / flat-ribbon connector** that mates with the IEEE-488 connector on the machine, and about 7 inches of **12-conductor ribbon** (14-conductor is standard — strip two conductors off).
- **Only `D0`–`D4` and GND are used.** The cable should leave the IEEE-488 connector going *up* and then loop down under the case for strain relief.
- The physical fact to keep: the machine's "IEEE-488" port is a **26-pin card edge connector**, and the article's Figure 1 is its pin assignment as seen from the front of the computer.

The bit mapping (Figures 5 and 6)

Data bits `D7…D0`	Decimal	Direction	Screen offset
`00000000`	0	center	0
`00000001`	1	north	−128
`00000010`	2	south	+128
`00000100`	4	east	+1
`00001000`	8	west	−1
`00000101`	5	north-east	−127
`00000110`	6	south-east	+129
`00001001`	9	north-west	−129
`00001010`	10	south-west	+127
`00010000`	16	**fire button**	(the program clears the screen)

The offsets are screen-memory deltas for the O1′s **128-character row** — a line is ±128, a character position ±1, so the diagonals are ±127/±129. Diagonals are *"the sum of the binary values (actually two simultaneous switch closures) — combinations such as 3 and 7 are not mechanically possible within the joystick."* Bits 5–7 are spare: *"If you have a joystick with three more switches, they could just as well be assigned to bit positions 5, 6, and 7 and 'sampled-in-software' accordingly."* The fire button has a quirk — *"As programmed, you MUST return to center position on the joystick to detect the closure of the 'Fire Button' ONLY"* — and the article warns that any switch wired in *"must have full ISOLATION from any source of external electrical power"* (its worked example of what not to do is a doorbell button: *"Zorch-City"*).

Reading it two ways

(a) High level. `CALL 0E154H` — the ROM's **Input Parallel Poll** driver (#8 above) — returns the switch-closure bit pattern in `A`. That is the whole interface for the article's "Etch-a-Sketch" checkout program, which walks a pointer around the 128×32 screen by adding the offsets above.

(b) Low level. `JOYSTIK.ASM` programs the 6821 directly:

```
porta0  equ 2900h  ; 6821 pia port-a data/direction register
porta1  equ 2901h  ; 6821 pia port-a control register
portb2  equ 2902h  ; 6821 pia port-b data/direction register
portb3  equ 2903h  ; 6821 pia port-b control register
flagpos equ 0ef08h ; ram/rom flag position address: 0=rom, 1=ram
```

`istick` floats both ports and then sets port A for input; `rstick` reads the joystick as `di` / `out 0` (ROM on, RAM off) / `xra a` / `sta flagpos` (tell the system the ROM is in) / `lda porta0` / `mov b,a` / `out 1` (RAM on, ROM off) / `mvi a,1` / `sta flagpos` / `ei` / `mov a,b` / `ani 01fh` (mask to `D0–D4`) / `sta sdata` / `ret`.

Two machine-level facts fall out of that listing:

- **`EF08H` is the ROM/RAM bank flag** the system reads: `0` = ROM, `1` = RAM. The routine has to keep it consistent with its own `OUT 0`/`OUT 1` bank switch, because it reads the port *while the ROM is banked in.*
- The mask is **`01FH`** — the port is read as a full byte and only `D0–D4` carry the switches.

Version dependence

The source is assembled at a **different origin per monitor ROM version** — `ver2` ⟶ `0EE80H`, `ver3` ⟶ `0ED80H`, `ver4` ⟶ `0EA80H` — because it calls into ROM addresses that move between revisions. The `.HEX` then has to be patched by hand for the machine's version (*"Change to 61"*, *"Change to CBE6"*, …), and the article says to press RESET and read the sign-on message if in doubt. For BIOS 1.4 the labels are `sdata` `EA80`, `rstick` `EA81`, `istick` `EA8B`, `end` `EAC8`, and the `.HEX` records load at `EA80`. This is the same version-dependence the first article flags from the other side (move CP/M down 1K on BIOS 1.3 or earlier).

The article is **part one of two**; the SPACEWAR game that uses the interface is promised for the next issue. Kelly Smith is with *Wizard of OsZ* and wrote "Monster" in the Companion's inaugural issue; he offered the SPACEWAR program (200 lines) and joystick hardware for a nominal fee (P.O. Box 964, Chatsworth, CA 91311).

6. The P3 edge connector — the pinout, and what the port actually implements

The pin-level map of P3 is in the Technical Manual extract the project already archives, §4.1.2 "IEEE-488 Pinouts" — the source `docs/drive-c-design.md` §7 and this document's §2.3 both build on. The manual explains why the connector is not the standard one:

> Because this port is used for more than just IEEE 488 signals, we've declined to use a standard IEEE connector.

Source: `research/osborne1/modem/o1techman-interface-extract.pdf`, PDF p. 2 = printed p. 14 (Figure 4.1.2), and Figure 4.1.1 on PDF p. 1 = printed p. 13 for the directions. **The scan has no text layer** — it is EPSON Scan output, so `pdftotext` returns nothing and the tables below were read off a 250 dpi render.

The connector is a **26-pin card edge**, read looking at the front of the machine, odd pins on one side and even on the other; **pins 25 and 26 are not used** ("provided for compatability"). The IEEE column is the standard 24-pin IEEE-488 assignment, kept here because it is what a GPIB cable or a GPIB device document will name.

IEEE-488	**P3**	Signal	PIA line (ROM)	O1 direction
1	**1**	DIO1	PA0	bidir
13	**2**	DIO5	PA4	bidir
2	**3**	DIO2	PA1	bidir
14	**4**	DIO6	PA5	bidir
3	**5**	DIO3	PA2	bidir
15	**6**	DIO7	PA6	bidir
4	**7**	DIO4	PA3	bidir
16	**8**	DIO8	PA7	bidir
5	**9**	EOI	PB3	O1 drives
17	**10**	REN	CB2	O1 drives
6	**11**	DAV	PB5	O1 drives
12	**12**	signal ground (DAV)	—	ground
7	**13**	NRFD	PB7	O1 drives
18	**14**	signal ground (NRFD)	—	ground
8	**15**	NDAC	PB6	**box may drive**
19	**16**	signal ground (NDAC)	—	ground
9	**17**	IFC	**CA2**	O1 drives
20	**18**	signal ground (IFC)	—	ground
10	**19**	SRQ	CA1	**input**
21	**20**	signal ground (SRQ)	—	ground
11	**21**	ATN	PB4	O1 drives
22	**22**	signal ground (ATN)	—	ground
12	**23**	cable shield + ground	—	ground
24	**24**	signal ground (logic)	—	ground
—	**25**	*unused*	—	—
—	**26**	*unused*	—	—

The PIA column is the ROM's own comment block (`rom144.asm` 2606–2715, quoted in `docs/drive-c-protocol.md` §2); the two sources agree line for line, which is worth saying because they were derived independently — one is a 1983 manual, the other a 1983 ROM listing.

Note what the PIA column does **not** contain: there is no DAV, NRFD, NDAC, EOI or IFC *input*. **The O1 cannot read any of the five handshake/control lines it drives.** Only SRQ (CA1) and NDAC (PB6, and only after CRB bit 2 is cleared) can be read back at all.

6.1 The port is not GPIB, and Figure 4.1.1 shows it

Figure 4.1.1 draws each line solid or dashed, and the split is the whole story:

Drawn	Lines	Meaning
solid	REN (out), ATN (in), SRQ (in), DATA0–7 (bidir)	implemented by the O1
dashed	IFC, NRFD, DAV, EOI, NDAC	*not implemented* by the ROM's driver set

The manual says as much in words — *"a subset of the complete IEEE specification"*, *"no provision has been made for controlling multiple devices"* — but the figure is the useful version, because it names the subset. **The three IEEE-488 handshake lines (DAV, NRFD, NDAC) are the ones missing**, which is exactly why nothing on this port can do an IEEE-488 three-wire handshake, and why every device on it invents its own clock. Drive C: uses CA2/IFC; the printer uses DAV as a strobe (§6.2).

ATN and SRQ are drawn as *inputs*, which is odd for a controller and correct for this machine: the O1 can only ever be told, not ask.

6.2 The same pins, as a Centronics printer port

Figure 4.1.5 (PDF p. 4 = printed p. 16) re-labels the same edge connector for the printer personality that SETUP selects (IOBYTE = BAT:, UR1:, UP1:, LPT:). It is the same wires, with four of them re-purposed:

P3	O1 name	Centronics pin	Centronics name
1	DIO1	2	data 0
2	DIO5	6	data 4
3	DIO2	3	data 1
4	DIO6	7	data 5
5	DIO3	4	data 2
6	DIO7	8	data 6
7	DIO4	5	data 3
8	DIO8	9	data 7
11	DAV	1	**out strobe**
12	ground	19	ground
15	NDAC	11	**busy**
16	ground	29	ground
19	SRQ	13	**select**

So `DAV` is the printer strobe, `NDAC` is *busy* and `SRQ` is *select* — and `NDAC` is the same line the Drive C transport leaves as an input as its optional "ready" (`docs/drive-c-protocol.md` §2, §7). **One pin, two meanings, chosen by which personality is installed.** Anything we put on P3 has to be harmless in both modes, which is a constraint `docs/drive-c-design.md` §2.3 now carries.

7. What this means for o1next

- The port is a **byte-wide parallel port on a 6821**, so a personality can implement it as such (`docs/virtual-peripherals-block-diagram.md`), and the **eight ROM drivers are the behaviour that has to be right**.
- **Address 0 is the machine's own** — anything else we place on the bus needs a different address.
- The port already has three quite different users in the archive: the ROM's driver set (this doc), a printer selected in SETUP (§4), and a dumb switch-closure box read through driver 8 (§5). Drive C: is a fourth, and the manual is explicit that it does **not** speak real GPIB on this connector (`docs/drive-c-protocol.md`).
- The driver table at **`013F–0156`** (eight `JP`s) is a *checkable* signature: present in stock ROMs, destroyed in OZROM 1E (`docs/o1-rom-variants.md`).
- **The pinout is solved** (§6): the Technical Manual's §4.1.2, with the ROM's comment block corroborating it line for line. So `hardware/drive-c-repro/` can be laid out against a documented connector rather than a measured one, and the one remaining electrical unknown on P3 is loading, not mapping.

Keyboard and printer

(not yet written)

Peripherals and storage

Drive C: the IEEE-488 RAM disk

Status: Solved. The host side of the contract is decoded and validated end to end; what is left is the box's internals, and a replacement does not need them.

Drive C is a battery-backed RAM disk on the Osborne 1's IEEE-488 edge connector, driven as a **parallel port** rather than as GPIB. The host side — the PIA line map, the frame, the two header bytes and the record address, the byte clock, and the DPB that sets the drive's size — is read out of `DCL.COM`'s disassembly and **confirmed against a trace of the real Drive C software running on the emulated machine** (§9): a model of the box built from this spec makes the 1983 loader complete (`QUICKPAC Drive C: loading has been completed`) and CP/M start copying to drive `C:` with PIP. What is *not* known is the box's internals — how it resolves `f(n)` for `n = 0, 1, 2` (§10 item 2), and how a print buffer interleaves with the pass-through printer (§10 item 3). Neither blocks a replacement: the contract in §7 is what the host exercises.

Sources: `research/drive_c/DRIVE_C.IMD` (the software), `research/drive_c/extracted/` (`dcl.com`, `dfd.spr`), `research/source-code/BIOS-144-source/rom144.asm` (the ROM's own account of the port), and `research/drive_c/Drive_C_Users_Manual.ocr.txt`. Serves #6 (the protocol), #84 (the reproduction), #22 (the IEEE-488 port the box hangs off) and #145 (the DPB read out of `DCL.COM`); `docs/drive-c-design.md` is the hardware + firmware spec built on it.

1. What the device is

A battery-backed RAM disk that hangs off the Osborne 1's **IEEE-488 edge connector** (P3) and duplicates that connector plus the Ext. Video connector on its own front panel, so the printer and a monitor can be plugged through it. It sold as 192 K and 384 K, in two parts: the unit (the RAM) and a software package (`DCL.COM` + `DFD.SPR`) that installs it as a CP/M drive and adds a print buffer. `DCL.COM` is the installer and carries the transport; `DFD.SPR` is a **substitute console command processor** that CP/M reloads on every warm boot in place of `CCP.COM` (`docs/drive-c-executive.md` §4).

The manual is explicit that this is not GPIB:

> **IEEE-488 COMMUNICATIONS PROTOCOL IS NOT SUPPORTED** ON THE Drive C PRINTER/HARD DISK/8088 EDGE CONNECTOR

The connector is used as a **parallel port** — the same trick the ROM itself uses for Centronics printers (§2). That is the single most important fact about the device: it is not a GPIB device and does not decode GPIB handshake semantics.

2. The electrical layer: the PIA, and what each line is

The host drives the box by bit-banging the motherboard's **MC6821 PIA at `2900–2903`** (bank 2). The ROM's own source documents what each pin is (`rom144.asm` §"IEEE-488 INTERFACE", lines 2606–2715), and it is the authority here — these are the vendor's comments, not an inference:

PIA	Line	GPIB name	Centronics use (`rom144.asm` 3250–3263)
PA0–PA7 (`2900`)	DIO1–DIO8	data	**data bus**
CA1 (`2901` bit 7)	SRQ	service request in	"low→high gates input data to port A"
CA2 (`2901` bits 5:3)	IFC	interface clear	**"going low indicates to device that we are busy"**
CB1	—	not used	—
CB2 (`2903` bits 5:3)	REN	remote enable	—
PB0 (`2902` bit 0)	enable data out		**`0` = data bus is output, `1` = input**
PB1	enable NDAC/NRFD		set to 1
PB2	enable EOI/DAV		set to 0
PB3	EOI		`0` output, `1` input
PB4	ATN		not used
PB5	DAV		**output strobe, active = 1**
PB6	NDAC		**`0` = printer busy, `1` = printer ready**
PB7	NRFD		not used

Two facts fall out, and both are load-bearing:

- **PB0 is the direction control for the data lines.** It is how the host says "I am writing" (`PB0=0`) or "you drive the bus" (`PB0=1`), and it is also the bit that distinguishes the two operations (§4).
- **All signals are active-low** — *"ALL SIGNALS ARE LOW ON THE IEEE BUS WHEN PIA REGISTER CONTAINS 1"* — and the O1 drives them through 7406 open-collector gates. A replacement must **release** lines, never drive them high.

Drive C's own use of the port (measured, §3) is a subset: port A for data, PB0 for direction, PB3/PB7 as framing, and **CA2 as the byte clock**.

3. The host's frame — measured

`DCL.COM` installs a relocatable block at `0x0600–0x07BE` (the file holds it at `0x1600–0x17BE`; the internal `JP 0757h`/`JP 0763h` are its *installed* addresses, which is what gives the relocation away). The transport routine is `0x06D5–0x07B4`; a helper at `0x07B5` writes port A and pulses CA2.

Every operation is one frame, and every frame re-initialises and tears down the PIA. A trace of the real software (10 195 PIA accesses, §9) shows this sequence, byte for byte:

```
DI                                  ; interrupts off for the frame
OUT (0),A                           ; bank 2 (the O1's own I/O bank)
CRA=32h ; PA<-FFh ; CRA=36h         ; port A = output
CRB=32h ; PB<-FFh ; CRB=36h         ; port B = output, CB2 (REN) high

PB<-08h                             ; frame open
PA<-h1   ; CA2 pulse                ; header byte 1
PB<-00h                             ; phase separator
PA<-h2   ; CA2 pulse                ; header byte 2
   [read frames only] CRA=32h ; PA<-00h ; CRA=36h   ; port A -> INPUT
PB<-88h  (write) | 81h (read)       ; the operation
   128 x {  PA<->byte ; CA2 pulse  }  ; the data phase

PB<-09h                             ; frame close
CRA=32h ; PA<-FFh ; CRA=36h         ; port A back to output
CRB=00h ; PB<-BFh ; CRB=04h ; PB<-02h ; port B: PB6 becomes an INPUT
OUT (1),A ; EI                      ; bank 1, interrupts on
```

Two details in the teardown matter: `PB<-BFh` is written with `CRB` bit 2 clear, so it lands in the **direction** register — **PB6 becomes an input** — and the final `PB<-02h` leaves the port in a resting state. So after every frame the box is *able* to drive PB6 (NDAC / "ready"), which is the one line the host could read for flow control.

4. Port B: the four states

Port B is written as a **state word**, not as individual handshake lines. Only four values are ever written during a frame, and each has a job:

Value	Bits set	Meaning
`08h`	PB3	**frame open** — EOI asserted low, data drivers disabled
`00h`	—	**phase separator** — between the two header bytes
`88h`	PB7, PB3	**write** — NRFD asserted, EOI low, and **PB0=0 so the host drives DIO**
`81h`	PB7, PB0	**read** — NRFD asserted, and **PB0=1 so the host releases DIO**
`09h`	PB3, PB0	**frame close** — EOI low, data drivers released

So 88h and 81h are the two operations, and the difference that matters is **PB0**: 88h = write to the box, 81h = read from the box. That is confirmed three ways — the op byte, the DDRA the host sets before the data phase, and the direction of the LDI in the transfer loop (§6).

5. Port A: the two header bytes, and the record address

Before the operation byte the host sends **two bytes on port A**, each clocked by a CA2 pulse. They are not a command and a length; they encode **the record address**, and the code says exactly how. The caller (DCL.COM 0x1122–0x115D, installed 0x0122) is entered with:

- HL = the caller's 128-byte buffer,
- DE = the **16-bit record address**,
- A = 0FFh to read, 0 to write;

and it computes

```
h1   = (DE >> 1) & 0FFh          ; sent first
n    = (DE >> 9) & 0FFh          ; high bits of the record address
h2   = f(n) | ((DE & 1) << 7)    ; sent second
```

where f(n) is the bank/size nibble the box wants. **f(n) is computed, not looked up**, and the code is short enough to quote — the installed routine at 0x0702:

```
    ld a,(051Ah)      ; 051Ah holds (DE&1)+1
    rrca
    and 80h
    ld b,a            ; b = (DE&1) << 7
    ld a,(0519h)      ; 0519h holds n
    cp 3
    jr c,+8
    sub 3
    rlca
    rlca
    or 3
    jr +2
    or 0ch
    or b              ; h2 = f(n) | ((DE&1)<<7)
```

so

n	0	1	2	3	4	5	6
f(n)	0Ch	0Ch	0Ch	03h	07h	0Bh	0Fh

h1 and n are not computed in the frame routine at all. The caller (0x0124–0x015F) shifts DE right by one and stores the **whole 16-bit result at 0x0518** — ld (0518h),hl — so 0x0518 receives (DE>>1)&0FFh = h1 while 0x0519 receives (DE>>1)>>8 = n. That is the cleanest confirmation that h1 and n are the low and high parts of *one* address, not two independent fields.

So (h1, h2) encodes DE>>1: h1 is its low eight bits, and its high bits ride in the low nibble of h2 through f. DE is the CP/M record address in 128-byte units, and the DPB says **SPT = 2** (§8) — so

`DE>>1` is the **track** and `DE&1` is the sector within it. The box's unit is a **256-byte track**, and bit 7 of `h2` is the sector.

The trace agrees with the code: across 163 frames of a boot-then-`PIP` run, `h2` took only `0Ch` and `8Ch`, never `Dh` or `Eh`. The consequence is not cosmetic — with `f` constant over `n = 0…2`, `(h1, h2)` no longer distinguishes those three banks, and §10 item 2 records that as the open question it is.

Measured examples from the trace: the install-time format writes `h1 = 00h…0Fh` with `h2 = 8Ch` (⟶ `DE = 1,3,5,…,31`) and `h1 = 00h…0Fh` with `h2 = 0Ch` (⟶ `DE = 0,2,4,…,30`); and the first operation is a read at `h1 = E8h`, `h2 = 83h` ⟶ `DE = 0x7D1`.

6. The byte clock, and the data phase

CA2 is the byte clock. The host writes `CRA=3Eh` then `CRA=36h` — CA2 low then high — once per byte, and the helper at `0x07B5` wraps "write port A, then pulse CA2". Nothing else is handshaken: **the transport never reads PB6 and never polls for ready**, so the box has to keep up. The loop is ~52 T-states per byte (about 13 µs at 4 MHz), and a read must be answered within ~6 T-states of the CA2 edge.

The order differs between the two directions, and that is the box's cue:

- **write:** the byte is placed on port A **first**, then CA2 pulses — the pulse says *latch it.*
- **read:** CA2 pulses **first**, then the host reads port A — the pulse says *present the next byte.* (A box that keeps the next byte on DIO and advances on each pulse satisfies this; one that only drives the bus after seeing the pulse has ~1.5 µs to do it.)

128 bytes are transferred per frame in both directions.

7. The device contract — what a replacement must do

This is the normative summary. A box that satisfies it will run `DCL.COM`, `DFD.SPR` and the rest of the software unchanged.

1. **Observe the PIA lines.** Port A (DIO1–8) is the data byte. Port B bits 0, 3, 5 and 7 carry direction and framing. CA2 is the byte clock. CB2/REN is held high and can be ignored.
2. **Decode the frame** from the port-B state words: `08h` opens, `00h` separates the header, `88h`/`81h` is the operation, `09h` closes.
3. **Capture the two header bytes** on their CA2 pulses (after `08h`, then after `00h`) and decode them to a 16-bit record address (§5).
4. **For `88h`,** latch 128 bytes from port A, one per CA2 pulse, and store them at that record.
5. **For `81h`,** present 128 bytes on port A, one per CA2 pulse.
6. **Release the data lines** whenever `PB0 = 1`; drive them only while `PB0 = 0`. Everything is open-collector: never drive a line high.
7. **Keep up with the clock** — there is no ready handshake in the transport. PB6 may be driven (high = ready) and is left as an input at the end of every frame, but nothing in the transport reads it.
8. **Survive the frame boundary.** Each frame is self-contained: init, header, data, teardown. A replacement may be stateless across frames.

8. Capacity — the DPB, the ceiling, and how `C:` appears

The box itself is a block store; the *filesystem* is CP/M's, and the geometry CP/M uses for `C:` comes from the **DPB that `DCL.COM` installs into the CBIOS**. That DPB is now in hand, and it is not a guess.

8.1 The DPB

`DCL.COM` is a .COM loaded at `0x100`, and its payload is relocated on install by `-0x1000` (the internal `JP 0757h` is the loop's own body at file `0x1757h`). At **file `0x142F` = installed `0x052F`** sits a bare 15-byte CP/M 2.2 DPB, and the payload's own pointer table names it (installed `0x0529` holds `2Fh 05h`):

```
file 0x142F:  02 00 04 0F 01 BF 00 7F 00 C0 00 00 00 00 00
```

Field	Bytes	Value	Meaning
`SPT`	`02 00`	2	128-byte records per track
`BSH`	`04`	4	block shift ⟶ 2 KiB blocks
`BLM`	`0F`	15	block mask (2 KiB − 1)
`EXM`	`01`	1	extent mask for `BLS = 2048`
`DSM`	`BF 00`	191	192 blocks × 2 KiB = **384 KiB**
`DRM`	`7F 00`	127	**128** directory entries
`AL0`	`C0`	`1100 0000b`	2 blocks reserved for the directory
`AL1`	`00`	0	—
`CKS`	`00 00`	0	no directory checking (fixed media)
`OFF`	`00 00`	0	no reserved tracks

Every field matches the vendor's own 384K column in the manual (§8.2), which is what makes this a reading rather than a guess. The `AL0` check is the nice one: `C0h` = the top two bits of the allocation vector = 2 blocks × 2 KiB = **4096 bytes** of directory, exactly the "Directory Space: 4096 bytes" the manual prints for the 384K unit.

8.2 The manual's own table — and the 192K unit

Drive C User Manual, §7-3 "Drive C SOFTWARE SPECIFICATIONS", printed p. 134 = PDF p. 142. The OCR in the archive mangles the column alignment, so this is read off a 300 dpi render. The three rows from *Reserved tracks* down are printed **once, centred across the two RAM-disk columns** — the 192K and 384K units share them — while the TurboPac column carries its own values:

CP/M parameter	192K unit	384K unit	TurboPac
Useful storage capacity	187 KB	376 KB	10, 20 or 33 MB
Block size	1K	2K	8K (can be modified)
Directory entries	64	128	512 per logical drive
Directory space	2048 bytes	4096 bytes	16384 bytes per drive
Logical sectors	1536	3072	varies with block size & capacity
Reserved tracks	**0**	**0**	612, landing area
Logical sectors/track	**2**	**2**	32
Bytes per sector	**128**	**128**	256
TPA RAM used	1152 B	1152 B	3172 B

Two independent confirmations fall out: **`SPT = 2`** matches the DPB exactly, and `1536 × 128 = 192 KiB` / `3072 × 128 = 384 KiB` match the two capacities.

The 192K unit's DPB is derived, not found — the archive holds the 384K unit's disk, and no literal `SPT=2, BSH=3, BLM=7, EXM=0, DSM=191, DRM=63, AL0=C0h` exists in any of the extracted files. It follows from the manual's column and the same arithmetic (192 × 1 KiB = 192 KiB; a 2048-byte directory is 2 × 1 KiB blocks):

Field	192K (derived)	384K (found)
`SPT`	2	2
`BSH/BLM/EXM`	3 / 7 / 0	4 / 15 / 1
`DSM`	**191**	**191**
`DRM`	63	127
`AL0`	`C0h`	`C0h`

Note that **`DSM` is 191 in both** — the two units hold the same *number* of blocks, just bigger ones. That is the design signature of the whole product.

8.3 The ceiling

Three separate limits apply, and the smallest one wins:

Limit	Where it comes from	Value
The protocol	`f(n)` is four bits: `((n-3)<<2)|3` has to stay inside the nibble, so `n ≤ 6`	**1792 tracks × 256 B = 448 KiB**
The DPB	`DSM` is 16 bits, so up to 65535 blocks	65535 × 16 KiB ≈ 1 GiB
CP/M 2.2	drive letters `A:`–`P:`	16 drives

The protocol bound is the interesting one, because it is the *host's own code* that imposes it: at `n = 7` the shift pushes bit 4 into the byte, the box's four-bit latch sees `03h`, and the frame collides with `n = 3`. So the shipped driver can address **at most 448 KiB**, and the largest unit the vendor sold (384 KiB) uses tracks `0…1535`, i.e. `n = 0…5` — the driver's `n = 6` branch is already headroom.

A replacement may therefore present up to 448 KiB and still be drivable by the unmodified 1983 software. Anything larger needs a new driver, which is outside what #84 is for.

And the manuals' own capacity figures reconcile with the DPB. The Osborne 1 book quotes 187 KB / 376 KB "useful"; the Executive book, for the same hardware, quotes 190 KB / 380 KB. Both are right, and the difference on the 384K unit is exactly `DFD.SPR`:

	192K unit	384K unit
Raw capacity	192 KB	384 KB
less the directory	−2 KB	−4 KB
= the Executive's "useful"	**190 KB**	**380 KB**
less `DFD.SPR` (Osborne 1 only)	−3 KB	−4 KB
= the Osborne 1's "useful"	**187 KB**	**376 KB**

380 KB is 190 blocks of 2K, which is `DSM = 191` (192 blocks) **minus the two directory blocks** — so the vendor's own figure *is* our DPB, reached independently. `docs/drive-c-executive.md` §3.

8.4 Drive letters, and what TurboPac actually is

- **`C:` is one drive and its letter is renamable.** `DCN.COM` ("DCN version 2.07", shipped on the disk) backs the `DCN x` command, and the manual warns that `COPY` requires the unit to be named `C:` while it runs. The DCL-era command is `DC1 A` (install and name it `A`); the older one was `DRIVEC A: C:`, which named the unit *and* the left floppy in one go. So the unit can take any of the 16 CP/M drive letters — **the limit is CP/M's, not the device's.**
- **TurboPac is not a bigger RAM disk — it is the *hard-disk controller*.** The manual is explicit, and it corrects a reasonable assumption:

> TurboPac software turns your Drive C into a hi-speed **Cache Buffer** for the Trantor hard disk. […] The 192K TurboPac system uses all of Drive C as a Cache Buffer. The 384K

> TurboPac system partitions Drive C into a 256K Cache Buffer and a 128K Fixed Print Buffer. **The hard disk attaches to the Drive C PRINTER/HARD DISK/8088 port** and no mechanical modifications are necessary.

So the Drive C unit is **also the host adapter for the Trantor Winchester**: the hard disk plugs into the Drive C's own pass-through connector, not into the O1. Two consequences: the pass-through connector in §2 is the *controller path* and not merely a duplicated printer port; and **a device that implements this protocol is not by itself a TurboPac** — that needs the hard-disk side too, which is a separate question (and one that overlaps #8).

8.5 What the trace shows about the format

At install the software writes `0xE5` (CP/M's blank byte) across two regions — `h1 = 00h…0Fh, h2 = 8Ch` and `h1 = 00h…0Fh, h2 = 0Ch`, i.e. `DE = 0…31` — then **reads each block back** to verify. Two regions, 16 blocks each, 128 bytes per block: 4 KiB, which is the *directory and boot area*, not the whole disk. The format is therefore incremental and continues as the disk is used. A later pass of writes touches `h1 = 18h, 19h, 20h, 28h` — the directory region again, at `DE` around `0x30–0x50` — which is `DCL` installing itself.

It happens once, not at every boot. The vendor's loader is guarded by a marker file on the unit: `DCQ T + n` skips its copy when **`DCLOADED.SYS`** is present (and creates it when it is not), and `DCL.COM` refuses a second install with `Driver is already installed`. A reboot — reset button included — therefore does not re-format the unit; forcing a reload means `ERA DCLOADED.SYS`, or `DCU E` to erase everything. The reproduction's persistence story rests on this: `docs/drive-c-design.md` §4.1.

9. Validation — the spec, run against the software

Two tools in `tools/research/` were written for this, and both are worth keeping:

- **`drive-c-trace.lua`** — write taps on `2900–2903`, logging every PIA access while the real software runs. Stable and reproducible: **10 195 accesses** on a 30-emulated-second run, which is the trace quoted throughout this doc. It is how the frame in §3 was *measured* rather than only read out of the disassembly — including the two facts the disassembly is easy to misread: that the header bytes go to **port A** (not port B), and that the read frame sets `DDRA=00h` before the operation.
- **`drive-c-box.lua`** — a **model of the box** built from this spec: it decodes the port-B states, reconstructs the record address, keeps 128-byte blocks, and answers reads. With it present, the real 1983 software **completes**:

```
QUICKPAC.  Loading Drive C:....
A>DCL
A>DCQ T + 1
A>PIP C:=A:*.*[WVOR]
COPYING -
AUTOST.COM
DCQ.DAT
```

i.e. `DCL` installs, QUICKPAC loads the driver, and **PIP starts writing to drive `C:`** — so CP/M sees the drive and the protocol is good enough for the shipped software. (`[WVOR]` includes `V` for verify, which reports a mismatch against this simplified model; the addressing is right, the model's `0xE5`-only initial contents are not.)

A MAME limitation, recorded because it cost time: a read tap that returns a *value* eventually segfaults MAME's Lua layer once the guest's resident driver starts polling (the run dies ~20 emulated seconds in; returning a constant instead keeps it alive). The trace tool is unaffected because it only writes taps. Treat the box model as a validation aid for short runs, not a harness.

10. What is still open

#6 closed on this document — its deliverable was the contract in §7 and the validation in §9, and both are done. What follows is the residue, kept because a reader deserves to know what is *not* known. None of it blocks a replacement; the first item below has moved to #84, which is where it belongs.

1. **The DPB — closed** (§8.1). It is `DCL.COM` file `0x142F`, installed `0x052F`, and it decodes field for field against the manual. What remains is *mechanical* rather than unknown: the payload's pointer table lists the DPB, the transfer buffer (`0x055A`) and two further addresses (`0x053F`, `0x0558`) with no obvious DPH among them, so **the DPH is presumably built at install time** rather than stored. That only matters if we ever want to re-create `DCL.COM`, which we do not.
2. **How the box resolves banks 0–2.** `f(n)` is `0Ch` for `n = 0, 1, 2` (§5), so `(h1, h2)` is *identical* for tracks 0, 256 and 512. Either the box ignores `n` below 3 — in which case the first 192 KiB aliases three ways and a 384 KiB unit could not work — or the nibble is not a bank select and the real bank comes from something not yet identified. This is now the sharpest question in the document. It is testable (write across the whole 384 KiB, read back) and it cannot be settled from software alone: the trace never reaches `n ≥ 1`, because the install-time format and `DCL`'s own install only touch tracks 0–40. **It does not block a reproduction** — a box that treats `f` exactly as the driver does behaves identically whatever `f` turns out to mean, provided the replacement presents 192 KiB or less. It is recorded as a **hardware-gated item on #84**, because it is the reproduction, not this document, that has to answer it.
3. **The print-buffer path — mostly closed.** The manuals give the budget (`docs/drive-c-executive.md` §5): a **fixed** buffer is carved out of the unit's own RAM disk, user-selectable from **16K to 128K in 16K steps**, and the 384K TurboPac configuration is stated outright as **256K cache + 128K fixed print buffer**. So the DPB in §8.1 is the *no-buffer* case and a unit with a fixed buffer presents a smaller `DSM` — confirmed by the manuals' own budget table. What is still not established is the spooling *protocol* — how the box interleaves buffer traffic with the pass-through printer connector — which only matters if a reproduction offers a buffer.
4. **`DFD.SPR` — closed: it is a replacement CCP, not the transport.** The vendor calls it "a substitute console command processor (CCP)" (appendix table), and that is what it is: a **replacement CCP**, loaded by CP/M on every `^C` and every program exit in place of the Osborne 1's `CCP.COM`, living in **user area 15** on the unit and required on logical drive `A:` during installation. The transport is `DCL.COM`'s (see *The transport lives in DCL.COM, not DFD.SPR*, below). A replacement

inherits `DFD.SPR` rather than reimplementing it, so no disassembly of it is needed. See `docs/drive-c-executive.md` §4.

Sources

- `research/drive_c/DRIVE_C.IMD` — the Drive C software disk (ImageDisk, 40-track FM 10×256); `research/drive_c/extracted/` holds the files.
- `research/source-code/BIOS-144-source/rom144.asm` — the ROM's IEEE-488 and Centronics line definitions (lines 2606–2715 and 3250–3269). The authority for §2.
- `research/drive_c/disasm/dcl.asm` — `z80dasm` of `dcl.com`; §3 and §5 are read from it (the resident block is file `0x1600–0x17BE`, installed `0x0600–0x07BE`). **The committed listing misaligns across the payload**: the installed code has to be read as bytes at file offset − `0x1000`, not from the disassembly, which is how §5′s `f(n)` and §8.1′s DPB were found.
- `research/osborne1/modem/o1techman-interface-extract.pdf` — Technical Manual chapter 4, including §4.1.2′s **P3 pinout**; transcribed in `docs/o1-ieee488-port.md` §6. Image-only scan: OCR it, do not `pdftotext` it.
- `research/drive_c/Drive_C_Users_Manual.ocr.txt` — the vendor's manual (OCR). §7-2 is transcribed in `docs/drive-c-design.md` §2.2, and §7-3 in §8.2 above; the OCR loses the spec table's column alignment, so read those pages from the PDF.
- `docs/o1-ieee488-port.md` — the port's device-level behaviour and the ROM's eight drivers.
- `tools/research/drive-c-trace.lua`, `tools/research/drive-c-box.lua` — the measurement and the model (§9).

Appendix — the files on the disk

File	Size	Role
DCL.COM	6656	"DRIVEC version 2.08" — driver installer/loader. Loads DFD.SPR, DCA/DCU. "Cannot install driver *Driverisalreadyinstalled*". Per the vendor's review it **replaces the earlier `DRIVEC.COM`**, and when the print buffer is switched on it copies `DCU.COM` into the unit
DFD.SPR	2560	**Substitute console command processor (CCP)**, page-relocatable SPR — the command layer, not the transport. Contains the command table (DIR ERA HELP REN SAVE TYPE USER) + BIOS hook code. The vendor describes it as "**a substitute console command processor (CCP)**" that is "**automatically loaded**"
DCA.COM	9984	"DCA version 1.85" — the **Archive-Retrieve** program per the vendor's own review (it "replaces both `DCARCH.COM` and `DCRET.COM`"); the `z80dasm` listing's own label for it, "assign/configuration", is the disassembly's, not the vendor's
DCU.COM	6272	"DCU version 1.40" — RAM-disk utility (erase all, directory, files/line config; generates DCUN.COM via SAVE 28)
DCQ.COM + DCQ.DAT	896+128	"DCQ version 1.22" QUICKPAC — fast boot-time loader; DCLOADED.SYS marker
DCQINS.COM	7296	QUICKPAC installer (writes boot track via DCL); text by David E. Price
AUTOEXEC.SUB / DCQAUT.COM	168	Auto-start file (`$$.SUB`) that runs QUICKPAC-autoload at boot

Author: Kenneth M. Toy, (c) 1983 Drive C. Also supports a print buffer ("Bad print buffer parameter, not loaded", "Not enough space for fixed print buffer"). Osborne-1 only: "cannot be used with the Executive".

The transport lives in DCL.COM, not DFD.SPR

`DFD.SPR` is **not** where the protocol lives — it is the command layer that drives it. `grep -E '290[0-3]h' dfd.asm` returns **nothing**: DFD.SPR never touches the PIA. DCL.COM holds all 22 PIA references. DFD.SPR's only port access is a single `OUT (0F1h),A` at `0x0803`, still unexplained (`F1h` is not a PIA address; likely the double-density board or a status port).

So the split is **DCL.COM = transport + installer**, **DFD.SPR = the command layer that drives it** (its own command table is DIR/ERA/HELP/REN/SAVE/TYPE/USER). How DFD.SPR reaches the transport is not established — it makes no direct PIA references, so it must be a patched vector or a relocation DCL performs at install time. That is a curiosity about the CCP rather than an item a replacement needs (§10 item 4, `docs/drive-c-executive.md` §4).

DFD.SPR's layout, and why the committed listing misleads (#6)

`research/drive_c/disasm/dfd.asm` is a `z80dasm` linear sweep from `0x100` over the whole 2560-byte file. Reading the file's own structure (a raw byte scan — the disassembly alone hides this) shows two large **zero-filled regions**, file offsets `0x003–0x0FF` (253 bytes) and `0x10C–0x17B` (112 bytes), plus a header/jump area and the command table:

File offset (= address − `0x100`)	Contents
`0x000–0x002`	`00 C6 07` — SPR header
`0x003–0x0FF`	**zeros** (253 bytes)
`0x100–0x10B`	`JP 0103h` / `JP 00FFh` + `STAT`
`0x10C–0x17B`	**zeros** (112 bytes)
`0x183` / `0x18B`	`COM` / `$$$ … SUB`
`0x1CC`	**command table**: `DIR ERA HELP REN SAVE TYPE USER`
`0x1F0`	command **vector table** (`0591h 0525h 0647h 0666h`)
`0x1F8`	`=_.:;<>`
`0x2A3`	`NO FILE`

Because the sweep starts at `0x100` and walks straight through those zero blocks, its `call sub_01aah` / `sub_01b1h` / `sub_01f5h` / `sub_020fh` … targets all land **inside the zeros** and are spurious — they are not entry points. So the command layer cannot be read off this listing; reading it would need a disassembly that starts at the code region and treats the zero blocks as data, and nothing in ##84 requires that (§10 item 4).

What *is* established from the bytes:

- DFD.SPR contains **no access to the PIA** `2900–2903h` (no `32 xx 29` stores) — confirming the split above.
- Its only I/O opcode is the single `OUT (0F1h),A` at `0803`.
- Its command layer is self-contained in the `0x180–0xB00` region, with a real command table at `0x1CC` and a vector table at `0x1F0` whose entries point *within the file* (`0x591`, `0x525`, `0x647`, `0x666`) — code, not relocation slots.

What the unit was like to use (contemporary reports)

Two letters in *The Portable Companion* #15 (Jan/Feb 1985, PDF p. 52) give the user side of the unit. That matters to the reproduction (#84): it has to be worth using, not merely protocol-correct.

> I've been using my **384K unit** for one month and it's turned my single density Osborne 1 into a totally new machine. … it takes **six seconds** to go from the editing no file menu in WordStar to the dot prompt in dBASE II. **Before I had Drive C: I had to boot up a new disk every time** I wanted to edit a program. That was minutes waiting while the Osborne booted up different files. … I had to shell out **$250 to Xerox for an exchange disk drive** two months ago. Now my **disk drives get 90% less wear** and I do 90% more work. — Martin Kilgore, South Pasadena, CA

And the reviewer, in the same issue, standing by a negative verdict:

> I still stand by what I wrote — **Drive C: is not a good buy for the average user**. … both of the readers who responded are doing precisely what I said the unit would be good at — **disk intensive operations for large parts of the day**. — Donald G. Krantz, Minneapolis, MN

So there was a **384K** configuration, and the disagreement was never about whether it worked — it was about who it was for. Extract: `research/technical-articles/TPC-Issue15-1985-pp52-drive-c-letters.pdf`.

The vendor's software set, from a contemporary review

The Portable Companion #14 (Nov/Dec 1984), printed pp. 41–45 = PDF pp. 43–47, carries **"Review of New Drive C: Loader & Archive Programs" by Larry Walker** — and, usefully for provenance, it is a **reprint from *FOGHORN* (First Osborne Group) 9/84**. It documents the second-generation Drive C: software, which is what this disk holds, and it names each program and its job:

Program	What the review says it is
`DCL.COM`	the loader "that makes the Drive C: unit accessible", replacing `DRIVEC.COM`; also turns the print buffer on, and then copies `DCU.COM` into the unit
`DFD.SPR`	"**A substitute console command processor (CCP)**, DFD.SPR is designed to make Drive C: work better and is **automatically loaded**"
`DCA.COM`	the **Archive-Retrieve** program, replacing both `DCARCH.COM` and `DCRET.COM`
`DCU.COM`	the print-buffer utility, loaded into the unit with the buffer; `DCU` gives a menu, `DCU E` erases every file
QuickPac: `DCQ.COM`, `DCQINS.COM`, `DCQ.DAT`, `DCQAUT.COM`	`DCQ.COM` "allows you to store a series of commands in SUBMIT fashion to load Drive C:, name it, copy the contents of a disk (or disks) into it, select a print buffer option, and run a program at the end of the process"; `DCQINS` is the menu-driven installer, writing the ASCII `DCQ.DAT`; startup becomes automatic by renaming `DCQAUT.COM` to **`AUTOST.COM`**
`DCN.COM`	"**automatically generated and loaded into Drive C: when the loader program is run**" — it backs the `DCN` rename command
`PRN.TST`	the print test file, sent to the printer at the end of installation

The archive lineage is spelled out too: `DCARCH.BAS` (needed MBASIC, "very slow") → `DCARCH.COM` (compiled, "bugs that arose when it was called upon to close files") → `DCA.COM`.

Command syntax of this generation

- `DC1 A` — install the unit and name it logical A. **This replaces `DRIVEC A: C:`**, which named the unit A and the left floppy C in one go.
- `DCN x` — rename the unit to logical name `x` at the CP/M prompt.
- `DC1 A16` … `DC1 A128` — as above plus a **fixed print buffer of 16, 32, 48, 64, 80, 96, 112 or 128K**. Substituting `P` installs the **dynamic** buffer — all otherwise unused memory, released back to programs as the buffer drains. Any other letter means no buffer is installed.
- `DCU` — buffer menu (stop/restart the dump, clear the buffer, report dynamic space); `DCU E` erases the unit.
- `DCA RB` — run Archive-Retrieve in **Retrieve** mode from drive B:.

Behaviour, and one gotcha that matters for emulation

- **A printer that is connected but switched off crashes the machine.** The review is explicit: *"If the printer is connected but OFF, the Loader will not work and the Osborne 1 will crash."* Cables must meet OCC's specification, and some printers (the reviewer's Daisywriter 2000) "look funny" to the unit — the fix being to disconnect the cable or leave the printer on. This is a property of the duplicated printer port, so a personality for #84 has to reproduce the failure and not only the success.
- **The reset button does not lose the unit's contents** — "only turning off the computer does that" — with the obvious advice to save first.
- **Archived files are stored contiguously as a single file** and are only readable through Retrieve; the directory shows one entry. A set spanning two disks names the second `filename.AR2`. The program warns when the target disk is not blank, shows its directory, and asks twice before erasing.
- **Speed**, as measured by the reviewer: `WSMSGS.OVR` (34K) retrieved into the unit in **10 seconds** versus **17** with PIP; a 272K WordStar + Spellguard set in 115 seconds. Single-density owners need two disk changes and take longer.
- **Cabling**: the unit mounts in the pocket below the right drive; the longer cable goes to the **IEEE-488 connector on the left side**, the shorter to the **External Video port** directly below the unit. Both O1 connectors are duplicated on the Drive C: panel (Centronics printer cable and shorting plug), and the unit has its own external video jack, so no adapter is needed.
- **Capacities sold**: 192K and 384K (the reviewer bought 384K).
- **Terms**: the new package was free to customers who bought after **31 December 1983**, $39 to earlier ones; the manual runs to nearly 150 pages.
- **BackPac** — a battery pack inside the keyboard enclosure that holds the unit's memory for **8–10 hours** after power is lost, "deliberately or through power failure". Announced but not shipping at the time of writing, manual still being written, target price **$150**.
- The same vendor resold **Trantor** hard disks at a discount. The Trantor advertisement itself (*The Portable Companion* #5, February/March 1983, PDF p. 28; extract `research/technical-articles/TPC-Issue05-1983-pp28-trantor-tsl-hard-disk-ad.pdf`) shows what the product was: the **TSL-5** and **TSL-10**, **5 MB $2995 / 10 MB $3395**, *"unpack your disk, **plug it into the IEEE port**, run a simple 5 minute installation procedure"*, supplied *"complete with BIOS, back-up, retrieve utilities and cable"*. So Trantor's Winchester is a **third IEEE-488-attached** storage device on this machine alongside Drive C — one more claimant on the port that #84 has to arbitrate.

Extract: `research/technical-articles/TPC-Issue14-1984-pp43-47-drive-c-loader-and-archive-programs.pdf`.

Disassembling

```
z80dasm -a -l -g 0x100 -o research/drive_c/disasm/dcl.asm research/drive_c/
extracted/dcl.com
z80dasm -a -l -g 0x100 -o research/drive_c/disasm/dfd.asm research/drive_c/
extracted/dfd.spr
```

z80dasm 1.2.0 (`brew install z80dasm`) — the same version and flags as the committed CoPower listings, so the two sets are directly comparable. It is a linear sweep, so it misaligns around data tables: DCL.COM has a data and jump-table region from roughly `0x0840`, and absolute `JP`/`CALL` targets that land there are table bytes or loader-patched relocation points, not code. (The `jp pe,00757h` / `jp pe,0763h` inside the data-transfer loop are the clearest example: `0x0757 + 0x1000` is the loop's own body, so the loader is adding a page offset when it installs the resident copy.) Read the listing with that in mind rather than trusting it end to end.

cpmtools `cpmls`/`cpmcp` with the `osb1sssd` diskdef worked perfectly, once the IMD was converted to raw with disk-analyse.

The Drive C reproduction: hw + firmware

Status: Solved as a design. The shape is settled and unblocked — the protocol it must speak is validated (`docs/drive-c-protocol.md`), the **DPB** that sets the drive's size is found (§8.1 there), and the **pin-level map of P3** is transcribed from the Technical Manual (`docs/o1-ieee488-port.md` §6). Nothing in the electrical or logical design is waiting on a measurement any more. Nothing is built either: what is left is a layout and firmware (§7).

Tracks: #84 (the device), #6 (the protocol), #22 (the port it hangs off) and #145 (the DPB its size comes from). Sibling of `hardware/keyboard-adapter/` and `hardware/wifi-comm-pac/` — a period device reproduced on modern parts, hanging off a port, reversible.

1. What we are building

A standalone replacement for the **Drive C** IEEE-488 RAM disk:

Element	Choice	Why
Bus	**P3**, the O1′s 26-pin IEEE-488 card edge	the device's own interface
Power	**P5**, the 10-pin video connector, via a passive shunt	EXMON precedent; keeps the machine unmodified
Compute	**RP2040** (a stock Pico)	bit-banging at the required rate is well inside PIO/GPIO
Storage	**external flash**, 512 KiB+	the largest Drive C was 384 KiB
Bonus	composite video out from the same shunt	the EXMON shunt already does this, passively

It is a **reproduction**, not an adapter: the machine sees a device on the port speaking the same protocol, so period software drives it unchanged. Look closely and you find an RP2040 where the original had a handful of TTL — which is the point.

1.1 The envelope, from the vendor's own spec sheet

§7-2 of the Drive C manual is a full specification sheet — and the Executive manual's §6-2 is the same page with one word changed. These are the numbers a reproduction should be measured against:

	Original (384K unit)
Dimensions, no cables	5.6 × 5.9 × 1.0 in (14 × 15 × 2.5 cm)
Weight / shipping weight	1 lb (0.45 kg) / 3 lb (1.4 kg)
Operating temperature	+5 to +60 °C (+40 to +140 °F)
Storage temperature	−20 to +100 °C
Power	11–14.5 VDC at 0.3 A max, **2.5 W nominal**
Latency	none — solid state, no moving parts
Max data throughput	**55,000 B/s** (445,000 b/s)
Random-sector access	≈40 µs (software access time)
Load a 32K `.COM` and run it	1.1 s max (1.0 s on the Executive)
TPA RAM used on the host	1152 B; +1024 B with the print buffer

The throughput figure is the one the firmware has to live inside: 55,000 B/s is **18 µs per byte sustained** (445,000 b/s over 8 bits), against a transport that clocks one byte per CA2 pulse in ~52 T-states (~13 µs, `drive-c-protocol.md` §6). §3.2′s timing has to stay within it.

And one design served two machines. The Osborne 1 and Executive variants differ only in software — the Executive's book describes an `ECL`/`ECN`/`ECU` loader for CP/M+ where the Osborne 1 uses `DCL`/`DCN`/`DCU` under CP/M 2.2 — while the mechanical, electrical and performance appendices are otherwise word for word. So this is **one design, not two**, and a board built to this spec serves either host. `docs/drive-c-executive.md` §1.

2. Electrical design

2.1 The bus

P3 is a 26-pin card edge (**pins 25/26 unused**) carrying the 6821′s lines (`docs/o1-memory-io-map.md` §6, `docs/o1-mainboard-schematic.md` sheet 9). The host drives it through **7406 open-collector drivers** with **3.3 K pull-ups** (UD6/UD8–UD11, RN7/RN11–RN16), and **all signals are active-low**. Three consequences, and they are the whole electrical design:

1. **Never drive a line high.** Every line is a wired-OR: the box pulls a line low or it releases it. A push-pull driver on this bus fights the O1.
2. **Level-shift.** The bus is 5 V; the RP2040 is 3.3 V and not 5 V tolerant. The same **74AHCT/HCT** conclusion that governs the interposer applies (`docs/research-gaps.md` gap 2): AHCT gates on the way in (5 V tolerant, TTL thresholds), and open-drain outputs (or AHCT with a series resistor) on the way out.

3. **Present a sane load.** The O1′s pull-ups are 3.3 K; the box should not add appreciably more capacitance or loading than one more TTL device on the bus.

Which P3 pin is which signal **is** in our material, and it never needed measuring: `docs/o1-ieee488-port.md` §6 transcribes §4.1.2 of the Technical Manual, and the ROM's own comment block agrees with it line for line. §2.3 gives the mapping for the lines this design actually touches.

2.2 Power

The EXMON precedent (`hardware/drive-c-repro/README.md`) takes power from **P5**, the 10-pin video connector, through a **passive shunt** — a Sullins MP-0100-10-DS-4 edge connector and a Hyko K688-DI sync combiner — and uses **+12 V**.

The budget is documented, not guessed. §7-2 of the Drive C manual gives the original unit's own supply requirement:

> Power Requirements: **11 to 14.5 VDC at 0.3 A max.** (2.5 Watts nominal, 384K unit). Source: Draws power from the **O1 EXT. VIDEO connector** or from the optional Drive C BackPac backup power supply.

So the vendor ran the whole 384K unit — 384 KiB of static RAM, the printer pass-through, all of it — off P5 at **0.3 A worst case**, from the same rail that feeds the CRT. An RP2040 plus flash is a small fraction of that (a Pico is ~50 mA at 3.3 V, well under 0.1 A at the rail after a buck). The shunt has to pass less than the original did, and the EXMON shunt is only a connector plus a sync combiner, so it does not change that arithmetic.

And the rail has room to spare, so the budget is closed (#354). P5′s +12 V is not separately limited — it arrives from `P7` pins 1/7 as plain wires (`1A2011-00` sheet 1 of 9; the machine's only fuse is the AC mains one), so the Astec AC8151′s **2.02 A** is the only limit. What sits on that rail is a 5" monochrome CRT (~1.0–1.2 A steady, ~2 A inrush), two floppy drive motors that never run together, and small mainboard supplies. The Drive C's 0.3 A — and any modern load, which draws milliamps — is lost in that. (The modem port is the opposite case: **limited**, by `R21` 22 Ω, #56.)

The shunt is a **mod**, even though it is passive and reversible, and must be documented as one — the same way the USB keyboard adapter's J6 bridge is.

2.3 Signals the box must actually watch, and where they are on P3

Per the protocol the box needs only five things, and `docs/o1-ieee488-port.md` §6 gives each of them a pin. The **P3** column is what the board is laid out against; the Technical Manual and the ROM agree on it.

Signal	PIA	**P3 pin**	Role in this design
DIO1–DIO8	PA0–PA7	**1,3,5,7,2,4,6,8**	the data byte
PB0	port B bit 0	*not on the connector*	direction: `0` = host drives, `1` = box drives — it gates the 7406s
EOI	PB3	**9**	low for the whole frame (`08h`, `88h`, `09h`) — frame framing
NRFD	PB7	**13**	high only in the operation byte (`88h`/`81h`) — what distinguishes it
IFC	**CA2**	**17**	**the byte clock** — one pulse per byte
NDAC	PB6	**15**	*optional* — the box may drive it high as "ready"; the transport never reads it

Three consequences a layout has to respect:

- **The data lines are interleaved across the edge** — `DIO1–4` on the odd pins `1,3,5,7` and `DIO5–8` on the even pins `2,4,6,8`. That is the card edge's own geometry, not a bit-order choice, and it is why a straight ribbon will not do.
- **Pin 15 is `NDAC` in this personality and *busy* in the printer one** (`01-ieee488-port.md` §6.2), while pin 11 (`DAV`) is the printer's strobe. A box that drives pin 15 has to stay harmless when the machine switches to the printer personality.
- **Pin 13 (`NRFD`) has to stay harmless as well.** The vendor's own printer-cable appendix (`docs/drive-c-executive.md` §6) warns that a cable carrying Centronics pin 12 (Paper Error) to pin 13 "will interfere with normal Drive C operation if connected and MUST be removed" — so the original Drive C was watching pin 13, and a box on the same line must not fight it.

PB1, PB2, PB4, PB5, CA1, CB1 and CB2 are not needed to satisfy the transport — though `PB1` and `PB2` are *enables* for the NDAC/NRFD and EOI/DAV drivers, so they gate lines this design reads (`NDAC`) and watches (`EOI`, `NRFD`).

3. Firmware — the Drive C personality

3.1 The frame state machine

The whole personality is a decoder for the frame in `drive-c-protocol.md` §3. It has five states and no timing assumptions beyond "keep up":

```
IDLE    PB=08h                    -> HEADER1      (clear the byte counter)
HEADER1 CA2 edge, DIO present     -> store h1
        PB=00h                    -> HEADER2
HEADER2 CA2 edge, DIO present     -> store h2; decode the record address
        PB=88h                    -> WRITE
        PB=81h                    -> READ
WRITE   CA2 edge                  -> latch DIO into record[n++]
READ    CA2 edge                  -> drive DIO with record[n++]
        n == 128, or PB=09h       -> IDLE
```

Notes that matter:

- **The header bytes arrive on port A, clocked by CA2** — not on port B. This is the single easiest thing to get wrong, and it is measured (`drive-c-protocol.md` §9).
- **The direction is PB0**, sampled from the operation byte (`88h` ⟶ host drives, `81h` ⟶ box drives). Release DIO whenever PB0 = 1.
- **The frame is self-contained.** No state has to survive a frame boundary, so a reset mid-frame costs nothing.
- **PB6** may be driven high continuously as "ready". Nothing in the transport reads it, so it is optional — but it is the only line the host leaves as an input after every frame, so driving it is the safe choice.

3.2 Timing

The host clocks a byte every **~52 T-states (~13 µs at 4 MHz)** and, on a read, samples DIO **~6 T-states (~1.5 µs)** after the CA2 edge. The safe design is to keep the next read byte **already on DIO** and advance on each CA2 edge, rather than to drive the bus on demand. An RP2040 does this comfortably in PIO or even in a tight GPIO loop: a 13 µs byte interval is 77 kHz, far inside either path's limit.

CA2 is sampled, not generated, so the box needs one edge-triggered input per byte. A PIO state machine with a wait-on-pin is the natural implementation, with the DMA feeding read data and draining write data.

3.3 Where the code lives

The personality is the same device as the interposer's IEEE-488 storage personality (#22) — one implementation, two hosts. `firmware/drive-c-gpib/` already exists as the home; this spec is what it implements.

4. Storage and geometry

Addressing. The host sends `DE`, the CP/M record address in 128-byte units, as the header pair, and `drive-c-protocol.md` §5 decodes it. `DE>>1` is the box's **track** (256 bytes), `DE&1` is the sector inside it, and `DE>>9` selects a 64 KiB bank. The box should decode the pair back to a flat byte offset — `DE * 128` — which is the simplest correct model and the one `drive-c-box.lua` uses.

Size: 384 KiB, and the ceiling is 448 KiB. The DPB is found (`drive-c-protocol.md` §8.1): `SPT = 2`, 2 KiB blocks, `DSM = 191` ⟶ **192 blocks = 384 KiB**; `DRM = 127` ⟶ 128 directory entries; `AL0 = C0h` ⟶ 4096 bytes of directory. That is the 384K unit the archived disk belongs to, and it

is what the box must present. The host's own code caps the addressable range at `n ≤ 6` ⟶ **448 KiB** (`drive-c-protocol.md` §8.3), so a **512 KiB flash** holds any Drive C that can ever be driven, with room for a copy of the original image beside it.

What the box does *not* do. It does not implement CP/M. The directory, allocation map and file structure live on the host side, in the CBIOS that `DCL.COM` patches in. The box is a paged 128-byte record store, and the format it sees (`0xE5` written across two regions and read back, `drive-c-protocol.md` §8.5) is CP/M formatting *its own* disk through the box — **once, at install, not at every boot** (§4.1).

One thing to leave open in the firmware. `f(n)` is identical for `n = 0, 1, 2` (`drive-c-protocol.md` §5), so the header alone cannot tell tracks 0, 256 and 512 apart. A reproduction that treats `f` exactly as the driver does is unaffected — it aliases exactly as much as the original does, if it aliases at all — but the firmware should key storage on the **raw `(h1, h2)` pair** rather than pre-decoding to an address, so that whichever answer turns out to be right can be implemented without touching the frame decoder.

4.1 Persistence: the format happens once, and the loader knows it

It is easy to read `drive-c-protocol.md` §8.5 as *"the unit is re-formatted on every boot"* and it is not true. The vendor's loader is **idempotent**, and the manual says so in as many words:

> `DCQ T + n` means: *"If there is a file on the Drive C unit named **`DCLOADED.SYS`**, then skip n lines in this file to the next command line. If no such file exists execute the next line, and create the `DCLOADED.SYS` file on the Drive C unit."* … *"This operation prevents QuickPac from copying all your files over from floppy to Drive C if you have already done so but are **forced to reload and reboot CP/M**."*

`DCL.COM` guards itself independently — it carries the strings `Cannot install driver` / `Driver is already installed` — and the contemporary review confirms it from the user's side: *"The reset button does not lose the unit's contents — only turning off the computer does that."* To force a re-load you must `ERA DCLOADED.SYS`, or `DCU E`, which erases the unit.

What that means for the reproduction. Nothing in the contract changes: the marker is an ordinary file in the CP/M directory, so it lives in records the box already stores, and **load/unload needs no special support** — QuickPac's copy-in and `DCA`'s Archive-Retrieve are bulk record traffic and nothing more.

The one divergence to decide, and the recommendation is to keep it: the original loses everything on power-off unless a **BackPac** battery is fitted (8–10 hours), and without one the marker goes too, so the next boot re-loads. Our storage is **flash**, so contents persist inherently — the behaviour of a BackPac-equipped unit. That is strictly better, costs nothing to implement, and means a working set loaded once stays loaded. A bare unit's power-off amnesia is deliberately **not** reproduced.

4.2 The blob: a raw CP/M image, so the tooling already exists

If the reproduction carries FAT32 storage (microSD or USB) and the image library (#20, #27) keeps a unit's contents as a file on it, **the blob format is the one thing that has to be chosen — and it does not need a new format.** The Drive C's memory *is* a CP/M 2.2 filesystem, so a plain raw image of it is already readable and writable by tooling this project uses.

The geometry is the DPB `DCL.COM` installs (`drive-c-protocol.md` §8.1), and it maps onto a `cpmtools` diskdef exactly:

```
diskdef drivec
  seclen 128
  tracks 1536
  sectrk 2
  blocksize 2048
  maxdir 128
  skew 0
  boottrk 0
  os 2.2
end
```

Checked, not asserted. Every DPB field falls out of that geometry and matches the bytes read out of `DCL.COM` — this is how `tools/cpmfs.py`'s new `drivec` format was verified:

DPB field	From the geometry	In `DCL.COM`
`SPT`	`sectrk` = 2	2
`BSH` / `BLM`	`blocksize` 2048 ⟶ 4 / 15	4 / 15
`EXM`	2048 B blocks, `DSM` ≤ 255 ⟶ 1	1
`DSM`	393 216 / 2048 − 1 = 191	191
`DRM`	`maxdir` 128 ⟶ 127	127
`AL0`	4096 B directory = 2 blocks ⟶ `C0h`	`C0h`
`OFF` / `CKS`	0 / 0	0 / 0

Why not `.IMD` or `.HFE`. Both encode *floppy* geometry — FM/MFM encoding, per-track sector IDs, the WD179x's world. The Drive C has no floppy controller: its "tracks" are 256-byte units of two 128-byte records addressed by `DE`. An IMD of it would mean inventing sector headers the hardware does not have. They are the right formats for drives `A:`/`B:` — which is exactly what #20 says — and the wrong one here.

Three consequences worth fixing now:

- **The file size *is* the geometry.** 393 216 bytes is the 384K unit; 196 608 is the 192K unit. A blob is self-describing by length, which is what a FAT32 library wants — no sidecar metadata, and a wrong-sized file is obviously wrong.

- **It has to be byte-exact, not file-level.** `DFD.SPR` lives in user area 15 and `DCLOADED.SYS` / `DRIVEC.SYS` are hidden system files, so an archive of the *files* (`.LBR`, `.ARK`, `.ZIP`) will not do — the image must round-trip whole.
- **The mount is #20/#27's business, not the box's.** The transfer is Pico-side, FAT32 ⟶ record store, triggered over the **mailbox port** (`docs/mailbox-protocol.md`, #155); `MOUNT.COM` only selects which blob. The box never parses FAT32, and nothing in §7's contract changes. This is also what makes "persistent Drive C" (#20) fall out for free: flash holds the records, and §4.1 says the loader will not re-format them.

What the DPB is, and how it was read. The DPB is the 15-byte CP/M 2.2 parameter block `DCL.COM` installs into the CBIOS; it sets the drive's size, and `drive-c-protocol.md` §8.1 reads it as bytes out of `DCL.COM` at file `0x142F` (installed `0x052F`). The whole round trip then passes through `cpmtools` as well, on an image built from the same diskdef — write, list, read back, byte-compare:

```
dd if=/dev/zero of=d.img bs=128 count=3072   # 393216 bytes; mkfs does not size
it
mkfs.cpm -f drivec d.img
cpmcp -T raw -f drivec d.img hello.txt 0:HELLO.TXT
cpmls  -T raw -f drivec d.img                # 0: hello.txt
cpmcp -T raw -f drivec d.img 0:HELLO.TXT hello.out
cmp hello.txt hello.out                      # byte-identical
```

Why `-T raw` is needed. `cpmls`/`cpmcp` hand libdsk a bare filename and let it autodetect the container — and libdsk 1.4.2's **IMD** driver answers for a file that is not an IMD. Its `imd_readto()` counts the read that hits EOF, so `imd_open()` then asks for one byte more than the file holds and returns `DSK_ERR_SYSERR`, which `dsk_strerror()` renders as `strerror(errno)` — *"No such file or directory"* on a file that is plainly there. IMD is probed **before** the raw drivers, so autodetection aborts there and never reaches the one that would have accepted the image. `-T raw` names that driver (`dc_posix`, the "Raw file driver") instead of asking for a guess; `-T logical` works equally well. `mkfs.cpm` needs no flag because it opens the file directly and never consults libdsk.

This is a 1.4.2-only bug, and it is already fixed upstream. libdsk **1.5.22** carries the same over-read in `imd_readto()` but reports it as `DSK_ERR_NOTME`, so the probe falls through to the raw driver instead of stopping. Measured against this image with both builds: 1.4.2's `dsk_open(…, NULL, …)` fails with `ENOENT`, a 1.5.22 build returns *no error*, and `-T imd` answers "Disc rejected by driver" rather than the bogus `ENOENT`. Homebrew ships **1.4.2** — upstream's *stable* release; 1.5.x is the development line — which is why the flag is what makes `cpmtools` usable here. The upstream ask is therefore a **backport** into the 1.4.x line, not a new fix: `tools/README.md` has the recipe, and **#157** is the report.

5. Coexistence: the pass-through printer, and address 0

Drive C duplicates the IEEE-488 connector on its front panel so a printer can plug through it, and the manual is explicit that **GPIB protocol is not supported** on that connector — it duplicates the *parallel port function*. So a reproduction has to decide what the pass-through connector is:

- **Minimum viable:** a mechanical/passive pass-through of the connector (a short ribbon), so a Centronics printer attached there still works with the machine's own parallel driver. That is what the original did electrically for the printer path, and it needs no firmware.
- **Full:** snoop the parallel traffic and spool it (the print buffer, `DCU`, `PRN.TST`). That is a separate feature and a separate issue.

Bus arbitration. Two devices claim this port on a real machine — Drive C itself and Trantor's hard disks (`docs/drive-c-protocol.md`) — and the O1′s **own address is 0**, so nothing else on the bus may use it (`docs/o1-ieee488-port.md` §1). A reproduction that only answers when it sees the Drive C frame pattern is compatible with both by construction: it stays released until it sees its own framing.

The **RT-60A is deliberately not in that list.** It is physically an IEEE-488 device, but this project implements its **ABI in software** rather than the hardware (`docs/rt60a-analysis.md`; #7 closed on exactly that basis), so no RT-60A sits on the bus in any configuration we build. It is a software contract, not a contender for the port, and the firmware does not need to arbitrate with it.

6. Test plan

1. **Protocol (done).** The frame, measured on the real software (`drive-c-protocol.md` §9).
2. **Box model (done, short runs).** `tools/research/drive-c-box.lua` makes the 1983 loader complete and PIP write to `C:`. Extend it into a proper harness — §7 item 3 — which is the fastest place to iterate on firmware logic before any hardware exists.
3. **DPB (done).** `drive-c-protocol.md` §8.1 — 192 blocks of 2 KiB, 128 directory entries, 4096 bytes of directory. Give the box model that geometry and a full `PIP` copy can be checked end to end.
4. **Hardware-in-the-loop.** With a real O1: `DCL`, then `DCQ`, then `PIP C:=A:*.*[WV]` and a byte-comparison of a file moved both ways — the same acceptance test #95 used for the floppy formats, and the one the software itself provides.

7. What is left

Everything this spec was waiting on has now been answered from material already in the archive, so this is a work list rather than a measurement list. The Executive manual comparison (`docs/drive-c-executive.md`) closed two more items and confirmed the hardware is a single design for both machines.

1. **Board layout — unblocked.** The P3 pinout is documented (§2.3, `docs/o1-ieee488-port.md` §6) and independently corroborated by the vendor's own printer-cable appendix (`docs/drive-c-executive.md` §6); the power budget is documented (§2.2); and the mechanical, electrical and performance envelope is in §1.1. Nothing on the board is waiting on a measurement.
2. **Bus loading — the one real electrical unknown.** How much additional load (capacitance, pull-up) the O1′s bus tolerates before its timings move. It is **not a design driver** and should be confirmed cheaply rather than designed around: the O1′s pull-ups are 3.3 K, the transport clocks a byte every ~13 µs, and one more TTL-class load — or the AHCT input the RP2040 side will present, which is lighter still — is a leakage-and-a-few-pF question at those edge rates.

What it decides is whether the board needs a **bus buffer at all**, so it is a confirmation step, not a blocker. A scope check with a spare TTL gate as the load settles it in an afternoon.

3. **Firmware against the model.** Turn `tools/research/drive-c-box.lua` into a proper harness (§6 item 2) and drive the §8.1 DPB geometry through it.
4. **The `f(n)` ambiguity** (`drive-c-protocol.md` §10 item 2). Not a blocker — keep the raw `(h1, h2)` key so the answer stays swappable (§4).
5. **TurboPac is out of scope for #84.** The unit is also the Trantor hard-disk host adapter (`drive-c-protocol.md` §8.4); that half needs the hard-disk side, which overlaps #8.
6. **A print buffer, if we offer one.** The manuals give its capacity and its fixed/dynamic split (`drive-c-executive.md` §5) — including that a fixed buffer is carved out of the RAM disk — but not the spooling *protocol*: how the box interleaves buffer traffic with the pass-through printer connector. Optional feature, not part of the RAM-disk contract.

Sources

- `docs/drive-c-protocol.md` — the protocol this implements (the contract), the DPB (§8.1) and the capacity ceiling (§8.3).
- `docs/drive-c-executive.md` — the Executive variant and the manuals' appendices: the envelope (§1.1), the print-buffer budget (§7 item 6) and the printer-cable corroboration of P3 (§2.3).
- `research/drive_c/Drive_C_Users_Manual.pdf` (OCR at `…ocr.txt`) — §7-2 hardware specifications (power, throughput, access time) and §7-3 CP/M parameters. Both spec pages are image-only with an OCR that loses the table alignment, so read them from the PDF: printed p. 133 = PDF p. 141, printed p. 134 = PDF p. 142.
- `research/osborne1/modem/o1techman-interface-extract.pdf` — §4.1.2 the **P3 pinout**, §4.1.1 the signal directions, §4.1.5 the same pins as a Centronics port; transcribed in `docs/o1-ieee488-port.md` §6.
- `docs/o1-ieee488-port.md` — the port's device-level behaviour, the ROM's eight drivers, and the joystick worked example.
- `docs/o1-memory-io-map.md` §6 — P3/P5 and the other connectors.
- `docs/o1-mainboard-schematic.md` sheet 9 — the IEEE-488 interface, the PIA and the open-collector drivers.
- `docs/interposer-survey.md` §5 — NODISKEMU and cbmSD, the IEEE-488 state machines to port.
- `hardware/drive-c-repro/README.md` — the device's place in the project, and the EXMON shunt precedent.

The Drive C's Executive variant

Status: Solved. The Executive manual is a direct port of the Osborne 1 one and the hardware is one design for both machines; the Executive's own loader software (`ECL`/`ECN`/`ECU`) is not in the archive.

The **Executive Drive C** is the same box as the Osborne 1's Drive C — same hardware, same IEEE-488 contract — sold against a different host. The Executive runs CP/M+, so its software set is `ECL`/`ECN`/`ECU` in place of `DCL`/`DCN`/`DCU` and there is **no `DFD.SPR`** to substitute for a CCP (§2, §4). The manual settles four things: the mechanical, electrical and performance appendices are word

for word the same, apart from the connector the unit draws power from and a 0.1 s better load time (§1); the CP/M parameter table carries the same numbers, which reconciles the two books' different "useful capacity" figures (§3); `DFD.SPR` is a replacement CCP rather than the transport, which is why the Executive needs none (§4); and the product's own printer-cable appendix corroborates the P3 pinout we transcribed from the Technical Manual, line for line (§6).

Source: `research/drive_c/Drive_C_Executive_Users_Manual.pdf`, OCR layer at `Drive_C_Executive_Users_Manual.ocr.txt`. Drive C:, 1690 65th Street, Emeryville CA. First edition July 1984, second printing 9/84; © 1984 *Drive C:, a joint venture*. 111 pages, image-only scan, OCR'd at 300 dpi per the `research/README.md` convention. The Osborne 1 manual already in the archive is `Drive_C_Users_Manual.pdf` — **second edition, June 1984**, same publisher.

Provenance: supplied by **Jim Rayburn** from a private Facebook group, posted 2024-01-06 — two posts, …/2063148960719887 and …/2023247321376718; recorded in `research/README.md`.

Bibliographic note: the cover is stamped **PRELIMINARY**, while every running header in the body reads "(final draft)" — so the PDF is a draft/review copy, not a production print. Its *content* is complete: the appendices are all present and internally consistent (§3).

1. The headline: the hardware is one design

§6-2 (Executive) and §7-2 (Osborne 1) are the same appendix. Verbatim, apart from one line:

Specification	Osborne 1 manual §7-2	Executive manual §6-2
Dimensions	5.6 × 5.9 × 1.0 in (14 × 15 × 2.5 cm)	identical
Weight	1 lb (0.45 kg)	identical
Shipping weight	3 lb (1.4 kg)	identical
Operating temperature	+5 to +60 °C (+40 to +140 °F)	identical
Storage temperature	−20 to +100 °C (−5 to +212 °F)	identical
Power	11 to 14.5 VDC at 0.3 A max (**2.5 W nominal, 384K unit**)	identical
Source	"the **O1** EXT. VIDEO connector"	"the **Executive** EXT. VIDEO connector"
Latency	none (solid state, no moving parts)	identical
Max data throughput	55,000 B/s (445,000 b/s)	identical
Random-sector access	≈40 µs (software access time)	identical
Load a 32K `.COM` and run it	1.1 s max	**1.0 s max**

The only substantive differences are the connector the unit draws power from and a 0.1 s better load time on the Executive — which is a host difference, not a Drive C one.

This is the useful result for #84. The Drive C is not two products; it is one box with two software personalities. The `docs/drive-c-protocol.md` contract is therefore the contract for *both*, and a replacement built to it serves either machine. It also means the P3 work in `docs/o1-ieee488-port.md` §6 transfers wholesale — the Executive's port is the same port.

The power figure is worth quoting precisely because it is the one number a reproduction has to beat: **2.5 W nominal** for the 384K unit, from a 11–14.5 V rail. A Pico plus flash plus SRAM is far inside that.

2. The software is not the same

The two manuals describe the same product in the same words, so the diff is clean. Everything below is a real difference:

	Osborne 1	Executive
Host operating system	CP/M 2.2	**CP/M+** (CP/M Plus), OCC BIOS 1.0/1.1
Loader	`DCL.COM`	**`ECL`**
Rename the unit	`DCN.COM`	**`ECN.COM`**
Utility program	`DCU.COM`	**`ECU.COM`**
QuickPac installer	`DCQINS.COM`	**`ECQINS`**
CCP replacement	**`DFD.SPR`** — 3K (192K) / 4K (384K)	**none**
Utility options	`S` space, **`O` optimize**, buffer, no-menu, `E` erase	`S` space, **`I` install/un-install**, buffer, **`D` file delete**, no-menu, `E` erase
System utilities appendix	COPY, SETUP, SYSGEN, **MOVCPM**	COPY, **COPYSYS**, SETUP
Floppy orientation	left / right	**upper / lower** (the Executive stacks its drives)
ZCPR appendix	§7-8	absent
CP/M tutorial section	Section 4	absent
Useful storage capacity	187 KB / 376 KB	**190 KB / 380 KB**
TPA RAM, no print buffer	1152 B	**768 B**
TPA RAM, with print buffer	2176 B	**786 B**
TPA RAM, with TurboPac	3172 B	**1024 B**
Disk overhead, 384K unit	8K min, 20K with options	**4K min, 16K with options**

Two of these are large enough to be worth stating plainly.

The Executive runs CP/M+, and that is why the driver differs at all. CP/M+ is a different system with a different BIOS and a different memory map, so the Executive's loader has to patch a different host. Everything the loader does is host-side; none of it touches the wire.

The Executive variant has no `DFD.SPR`, and uses a fraction of the TPA RAM. Both follow from the first point, and §4 explains why.

3. The CP/M parameters are identical — and the capacity figures reconcile
§6-3 (Executive, headed "CP/M+ PARAMETERS") and §7-3 (Osborne 1, "CP/M PARAMETERS") carry the same table, with the same numbers:

Parameter	192K unit	384K unit	TurboPac
Useful storage capacity	190 KB	380 KB	10, 20 or 33 MB
Block size	1K	**2K**	8K (modifiable)
Directory entries	64	**128**	512 per logical drive
Directory space	2048 B	**4096 B**	16384 B per drive
Logical sectors	1536	3072	varies
Reserved tracks	**0**	**0**	612, landing area
Logical sectors / track	**2**	**2**	32
Bytes per sector	**128**	**128**	256

The three rows from *Reserved tracks* down are printed **once, centred across the two RAM-disk columns** — the 192K and 384K units share them — while the TurboPac column carries its own values. The OCR loses that alignment; read column-wise it would put 32 sectors per track and 256-byte sectors on the 384K unit, which the DPB `DCL.COM` installs refutes (`SPT = 2`, 128-byte records, `docs/drive-c-protocol.md` §8.1).

The 384K column is our reverse-engineered DPB, field for field. Block size 2K is `BSH = 4`; 128 directory entries is `DRM = 127`; 4096 bytes of directory is `AL0 = C0h` (two 2K blocks). And the capacity falls out exactly: `DSM = 191` means **192 blocks of 2K = 384 KB total**, of which the directory takes 2, leaving **190 blocks = 380 KB usable** — the Executive's own figure.

That in turn explains why the two manuals quote *different* "useful" capacities for the same unit, which had looked like a documentation error:

	192K unit	384K unit
Raw capacity	192 KB	384 KB
less the directory	−2 KB	−4 KB
= Executive's "useful"	**190 KB**	**380 KB**
less `DFD.SPR`	−3 KB	−4 KB
= Osborne 1's "useful"	**187 KB**	**376 KB**

Both books are right. The Osborne 1 figure is *net of the CCP replacement the Osborne 1 needs and the Executive does not* — which is a neat independent confirmation of §4.

4. `DFD.SPR` is a replacement CCP, not the transport

`DFD.SPR` is a **substitute console command processor** — a CP/M program that CP/M reloads on every warm boot in place of the Osborne 1's `CCP.COM`. It is not the transport and it is not device firmware: the transport is `DCL.COM`'s (`docs/drive-c-protocol.md` §3). The Osborne 1 manual's software appendix says so in the vendor's own words, and the `research/README.md` row for the *FOGHORN* review says the same.

The Osborne 1 manual settles it, in the vendor's own words and in more detail than the review had:

> The `DFD.SPR` file is absolutely necessary to Drive C operation. It's located in **user area 15**. DO NOT ATTEMPT TO ERASE `DFD.SPR` FROM USER AREA 15. This file is loaded each time you type `^C` and each time a program exits to CP/M. … The Drive C software automatically loads the `DFD.SPR` file **to replace the O1 Digital Research CCP** which would normally be loaded from the drive A: floppy disk after each `^C`.

So the picture is:

- **`DCL.COM` is the loader and carries the transport.** It announces itself as `DRIVEC version 2.08`, carries `DCN version 2.07` inside it, and creates `DCN.COM` on the unit when it runs.
- **`DFD.SPR` is the command processor** that CP/M reloads on every warm boot in place of `CCP.COM`. It is a *CP/M program*, not device firmware — which is exactly why its committed listing "sweeps through zero blocks": it is a CCP image, not a driver.
- **It must be present on logical drive `A:` during initial installation**, and thereafter lives on the unit in user area 15. Its name "has no significance, but `DCL` always checks for its existence each time `DCL` is run."
- **It costs 3K on a 192K unit and 4K on a 384K unit** — a number the review did not give, and the one that makes §3's capacity arithmetic close.

Why the Executive has none of this. The Executive runs CP/M+, whose CCP is a different animal; the O1's trick of substituting a custom CCP on warm boot has no counterpart there. That is the whole reason the Executive variant is 4K lighter on the unit and uses 384 fewer bytes of TPA without a print buffer (768 vs 1152), and 1390 fewer with one (786 vs 2176).

What this changes for #84. A replacement does not need to reimplement `DFD.SPR`: it inherits it. The CCP that ships with the product already talks to whatever driver is installed, so a box that satisfies `drive-c-protocol.md` §7 gets the command layer for free, and no disassembly of `DFD.SPR` is needed — `docs/drive-c-protocol.md` §10 item 4 records that.

5. The disk and RAM budget, from the manuals

§7-4 (Osborne 1) and §6-4 (Executive) give the unit's overhead directly. This is the passage that answers whether a **fixed print buffer** costs RAM-disk space — it does:

	Osborne 1		Executive	
Function	192K	384K	192K	384K
Directory space	2K	4K	2K	4K
`DFD.SPR`	3K	4K	—	—
`DCN.COM` / `ECN.COM`	1K opt	2K opt	1K opt	2K opt
`DCU.COM` / `ECU.COM`	10K opt	10K opt	10K opt	10K opt
minimum	**5K**	**8K**	**2K**	**4K**
with options	**16K**	**20K**	**13K**	**16K**

`DCL.COM` also creates a second system file the disk-space table does not list — the manual's screen illustration shows `DCLOADED.SYS` and `DRIVEC.SYS` on the unit, and notes that `DRIVEC.SYS` is "a SYSTEM file and will not be displayed by DIR. To check for the `DRIVEC.SYS` file, use XDIR or STAT." (Those two names come from a poor OCR of a screen dump, so treat the spelling as provisional; what is solid is that there are two, they are hidden system files, and `DCL` creates them.)

RAM. The O1 needs **1152 bytes** of the 64K, plus **1024** more if the print buffer is installed — and the buffer's RAM "is not used unless you install the Buffer at the time you run `DCL.COM`". The manual even names the consequence: applications that use all of the TPA (it cites DeskTop Accountant) may need the print buffer removed to load.

Print buffer, in the unit's own memory:

- Fixed Buffer Capacity: user-selectable **16K to 128K in 16K steps**.
- Dynamic Buffer Capacity: up to the maximum space available on the unit.
- TurboPac spool buffer: **128K fixed on the 384K unit, not installed on the 192K unit**.

That last line is the 192K/384K TurboPac split, and it is also the explanation for the print buffer's disk cost: a fixed buffer is carved out of the RAM disk, so a 384K unit running TurboPac is **256K cache + 128K fixed print buffer** — which is precisely what the Osborne 1 manual says in its feature list.

6. The printer connector is P3, re-presented — and it checks our pinout

Both manuals carry the same appendix (§7-5 / §6-5) giving the cable from the unit's **PRINTER/ HARD DISK/8088** connector to a Centronics printer. That connector is a **26-pin card edge** —

the same edge as P3 — and its pin assignments match our transcribed P3 table (`docs/o1-ieee488-port.md` §6, §6.2) **line for line**:

Drive C 26-pin edge	Signal (manual)	Our P3 table	Centronics
1, 3, 5, 7	data bits 0–3	`DIO1`–`DIO4` (odd pins)	2, 3, 4, 5
2, 4, 6, 8	data bits 4–7	`DIO5`–`DIO8` (even pins)	6, 7, 8, 9
11	Output Strobe	`DAV`	1
13	**NRFD**	`NRFD`	(12, PE — must be cut)
15	Printer Busy	**`NDAC`**	11
19	SRQ	`SRQ`	13 (SLCT OUT)
12, 14, 16, 18, 20, 22, 23, 24	ground	the same eight grounds	19–30

Four things this buys us:

- **The interleave is confirmed by a second, independent source.** Odd pins are data bits 0–3, even pins are 4–7 — the vendor's own cable table, not just Figure 4.1.2.
- **The ground set is confirmed exactly** — pins 12, 14, 16, 18, 20, 22, 23, 24, the same eight our table lists.
- **The dual-purpose pin 15 is confirmed.** It is `NDAC` on the bus and *Printer Busy* in the printer personality — precisely the "one pin, two meanings" constraint `docs/o1-ieee488-port.md` §6.2 already recorded, now corroborated by the product that has to live with it.
- **The Drive C watches pin 13 (`NRFD`).** The manual is emphatic that a cable carrying Centronics pin 12 (Paper Error) to pin 13 "will interfere with normal Drive C operation if connected and MUST be removed". So the box must leave `NRFD` harmless as well as `NDAC` — a constraint for `docs/drive-c-design.md` §2.3.

One discrepancy between the two tables. Our P3 table has pin 10 as `REN` (`CB2`), while the Drive C's cable table lists pin 10 as **ground** (tied to Centronics pin 30). Either the Drive C's cable grounds a line it does not need, or the two connectors are not bit-identical on that pin. It does not affect anything a replacement has to do — `REN` is unused by every personality we care about — but the two tables disagree and that should be noted rather than assumed away.

7. What the Executive manual does *not* have

Worth listing, because absence is evidence about how the product was maintained:

- **No `DFD.SPR`**, and no equivalent. CP/M+ needs no CCP substitution (§4).
- **No ZCPR appendix.** The O1 manual's §7-8 covers running ZCPR with Drive C; the Executive's appendix list stops at §6-6.
- **No CP/M tutorial.** The O1 manual's Section 4 teaches CP/M; the Executive manual renumbers everything up by one and drops it, so its Section 4 is the RAM-disk/print-buffer/utility section.

- **No MOVCPM rule.** The O1′s §7-1 warns that `MOVCPM` "must be run BEFORE running the Drive C Loader, `DCL.COM`" or it reports a "Synchronization Error". The Executive's §6-1 has no equivalent — CP/M+ is configured with `SETUP`, and `GENCPM`-style rebuilding is a different mechanism.
- **No fourth system utility is named.** The Executive's §6-1 opens "There are four programs supplied with your Executive" and then documents three: COPY, COPYSYS and SETUP. The O1′s §7-1 names all four of its own (COPY, SETUP, SYSGEN, MOVCPM). Whether the Executive's fourth was dropped from the draft or simply never written is not determinable from this copy.

8. What this changes

For #84, the Executive variant is not a second design to build. It is the same box with a different loader, and the contract in `docs/drive-c-protocol.md` is the contract for both. The reproduction target does not change.

It removes work rather than adding it. `DFD.SPR` is a CCP, it ships with the product, and a replacement inherits it — so no disassembly of it is needed (§4).

It closes the fixed-print-buffer question. A fixed buffer is carved out of the unit's RAM disk, and the manuals give the budget table (§5). The 384K TurboPac split — 256K cache + 128K fixed print buffer — is stated outright.

It corroborates the P3 pinout from a second vendor document (§6), including the interleave, the ground set and the dual-purpose pin 15.

It supplies the mechanical, electrical and performance envelope (§1): 5.6 × 5.9 × 1.0 in, 1 lb, 11–14.5 VDC at 0.3 A max and **2.5 W nominal** for the 384K unit, 55,000 B/s throughput, ≈40 µs random-sector access. Those are now targets rather than unknowns in `docs/drive-c-design.md`.

What is still open

1. **How the box resolves banks 0–2** (`drive-c-protocol.md` §10 item 2) — unchanged by any of this, still the sharpest question, still not a blocker.
2. **Bus loading** (`drive-c-design.md` §7 item 2) — unchanged.
3. **The Executive driver software.** The manual tells us what the Executive's loader is called (`ECL`) and what it does, but we do not have it. If one surfaces, the interesting comparison is *why the Executive driver needs 768 bytes of TPA where the Osborne 1′s needs 1152* — the answer is in how each patches its host, and it would tell us how much of the driver is host adaptation versus device work.
4. **`DCLOADED.SYS` and `DRIVEC.SYS`** (§5) — two hidden system files that `DCL` creates on the unit and the disk-space table does not itemise. Their names come from a poor OCR of a screen illustration, so even the spelling is provisional.
5. **Pin 10** (§6) — `REN` in our P3 table, ground in the Drive C's cable table.

Sources

- `research/drive_c/Drive_C_Executive_Users_Manual.pdf` and `Drive_C_Executive_Users_Manual.ocr.txt` — the Executive manual. First edition July 1984, second printing 9/84; © 1984 Drive C:, a joint venture. 111 pages, image-only; OCR'd at 300

dpi per the `research/README.md` convention. Owner-supplied local file, 2026-10-11; source URL not yet recorded.

- `research/drive_c/Drive_C_Users_Manual.pdf` and `…ocr.txt` — the Osborne 1 manual, second edition June 1984. The source of §4, §5 and most of §3.
- `docs/drive-c-protocol.md` §3 (the frame), §8 (the DPB and the ceiling), §10 (what is still open).
- `docs/o1-ieee488-port.md` §6/§6.2 — the P3 pinout §6 corroborates.
- `research/drive_c/extracted/dcl.com` — `DRIVEC version 2.08`, `DCN version 2.07`, and the `DFD SPR` / `DCN COM` / `DCU COM` name literals quoted in §4.

The hard-disk products

Status: Solved. As a product analysis: the OCC1 is Australian Computer & Telecommunications' 1982 Osborne 1 hard disk, and the mass-storage decision went the other way — emulate the **Adaptec ACB-4000** inside the MD-10, not the ACT (`docs/harddisk-decision.md`). The ACT's hardware interface below is read out of its own BIOS image. What remains is archival only: no ACT board photograph or schematic has surfaced (#8).

What this is. The ACT unit is a hard disk for the Osborne 1 that attaches through the Z80 socket, and it is one of **two unrelated** Osborne 1 hard-disk products in the archive — the other is the Media Distributing / Adaptec **MD-10** (**Two products, not one**, below). The archive holds its driver disk and nothing else, so the ACT is documented here as prior art: it validates the Z80-interposer concept on this exact machine, and it is *not* the controller the project emulates.

Source: research/harddisk/OCC1_HARDDISK.IMD (Osborne 1 SSDD, FM 10x256). Extracted to research/harddisk/extracted/ via disk-analyse + cpmtools (`osb1sssd`).

What this is

The Osborne 1 hard disk product from **Australian Computer & Telecommunications (ACT), (c) 1982** (string in BOOTHd.COM). This is the "winchester style" interface — and, significantly for us, it is one of the known **Z80-interposer** designs: the host-side hardware plugged into the Osborne's Z80 socket.

Two products, not one

The archive drop (#71) added a **second, unrelated** Osborne 1 hard-disk product, so "the OCC1 hard disk" is a family, not a single thing:

	ACT (this doc)	Media Distributing / Adaptec (**the MD-10**)
Source	`research/harddisk/OCC1_HARDDISK.IMD`	`research/adaptec/*.img`
Vendor string	`AUSTRALIAN COMPUTER & TELECOMMUNICATIONS COPYRIGHT (C) 1982`	`Copyright (c) September 1983, Media Distributing`
Versions	`Version 31.05.82`, `Version 9-02-81`	`HARD version 1.93`, `PORTCHNG version 1.92`, `PREP version 1.93`
Controller	its own; prompts `Enter disk controller address (`	Adaptec ACB-4000; prompts `Enter starting port address in hex / must be on a 4 port boundary`
BIOS	`HARDBIOS.HEX`, overlaid by `LOADBIOS.COM`	`HARDBIOS.SPR`, `HARDBDOS.SPR`, `HARDCCP.SPR`, `HD00BIOS.SPR`, `HD04BIOS.SPR`
Other utilities	`MOVCPM10/5/F`, `DISKEDIT`, `DISKTEST`, `RESTORE`, `SAVEFILE`	`MD10.COM`, `MD20.COM`, `PREP.COM`, `DRIVETBL.DAT`, `PORTCHNG.COM`

The second product is identified: it is the Mega-byte MD-10. Mega-byte Inc.'s advertisement in *The Portable Companion* #5 (February/March 1983, PDF p. 25; `research/technical-articles/TPC-Issue05-1983-pp25-mega-byte-md10-hard-disk-ad.pdf`) sells the **MD-10** — 10 MB formatted for $2295, "20 and 44 megabyte Winchesters also available" — and its footnote reads *"MD-10 is a registered trademark of **Media Distributing**."* The disk in `research/adaptec/` is copyright Media Distributing (September 1983) and ships `MD10.COM`/`MD20.COM`, matching the 10 and 20 MB models, so the "Adaptec" disk is **MD-10 software**. Two details from the ad bear on the interface question below: the MD-10 "does not tie up existing OSBORNE ports" and is "significantly faster than other winchester subsystems **which use the IEEE port**" — so it is **not** an IEEE-488 device, consistent with a controller that claims its own addresses (hence `PORTCHNG.COM`) rather than borrowing the serial/video ports. Its own parts list — "drive controller, case, power supply, cabling, **Z-80 interface**" — points at the same class of host attachment as the ACT unit: a **Z80-socket or dedicated-port** design, not the IEEE bus.

Both are **port-relocatable** controllers: the ACT disk asks for a "disk controller address", the Adaptec disk for a "starting port address … on a 4 port boundary", and it ships `PORTCHNG.COM` to change it. That is consistent with the rest of the O1, whose I/O decode only looks at the low address lines.

That is also what makes the port claim resolvable: `0x2A00` and `0x2C01` are the console ACIA and the video PIA on this machine (`docs/o1-memory-io-map.md` §3), so the ACT adapter is mapped elsewhere — the **IEEE-488 PIA at `2900–2903`**, read out of the BIOS below.

So there are **three** routes to more storage than 2 × 182 KB — the two hard disks here, and Drive C (`docs/drive-c-protocol.md`) — and only Drive C is buildable without unobtainable hardware (#84).

Files

File	Role
HARDBIOS.HEX	Intel HEX of a full replacement BIOS, loads at 0xDF00-0xEB00 (17-jump BIOS entry table at 0xDF00)
LOADBIOS.COM	Overlays HARDBIOS.HEX onto the running BIOS ("Your BIOS has been overlayed"; validates 17 JMPs, checksums, warm-boot vector)
BOOTHd.COM	Hard-disk boot ("ERROR DURING BOOTING TO HARD DISK")
MOVCPM10/5/F.COM	Stock Digital Research `MOVCPM.COM` builds — three relocations of CP/M under the new BIOS, **not** 10 MB / 5 MB volume sizes
DISKEDIT.COM, DISKTEST.COM	Winchester utilities
RESTORE.COM, SAVEFILE.COM	Backup/restore to floppy
AUTOST.COM, SYSGEN.COM, PIP.COM, XDIR.COM	Standard

Hardware interface (from HARDBIOS.HEX disassembly)

Read by disassembling `HARDBIOS.HEX` (`z80dasm`, `org 0xDF00`, `0xDF00–0xEB00`). The interface is **not** the `0x2A00`/`0x2C01` pair those addresses would suggest — they are the console ACIA and the video PIA (`docs/o1-memory-io-map.md` §3) — and the image drives exactly one device. `docs/harddisk-decision.md` §3 has the same reading in the decision's own words.

- Same Osborne bank discipline as Drive C: **OUT (0x00) = bank 2 (IO), OUT (0x01) = bank 1 (RAM)**, DI/EI bracketed (subroutines at 0xEA6A / 0xEA75 wrap all IO accesses).
- **The disk transport is the IEEE-488 PIA at `2900–2903`**, bit-banged: `set 3,(02902h)` / `res 3,(02902h)` is a **PB3 strobe** around `ld a,(02900h)` (read) and `ld (02900h),a` (write), with `cpl` for active-low. It is the only device routine in the image, reached from the disk path READ (`E32A`) ⟶ `sub_e448h` ⟶ `E475` ⟶ `sub_e520h` ⟶ `sub_e58fh` (`E58F`).
- **`0x2A00`/`0x2C01` are not the disk.** `0x2A00`/`0x2A01` are the console **ACIA** (`0x2A00` status read / control write, `0x2A01` RX/TX — the console helpers `sub_e992h`/`sub_e99bh` test bit 1 = TDRE, bit

0 = RDRF) and `0x2C01` is the **video PIA** (`docs/o1-memory-io-map.md` §3). The image contains **no floppy-controller references at all**: it is a hard-disk-only BIOS.

- The "indirect `OUT (C),A`, target TBD" is resolved: `C` comes from `[0xEF08]`, which the bank helpers set to `0` or `1` — it is the **bank restore** on exit, not a disk port.
- The BIOS preserves a full register environment (pushes IX/IY/HL/DE/ BC/AF, switches to a private stack at 0xEFC1) around hard-disk calls — it hooks deep into the 17-entry BIOS jump table as a full BIOS replacement, not just a disk-driver wedge.

Comparison with Drive C

	Drive C (RAM disk)	ACT hard disk
Physical interface	IEEE-488 port (6821 PIA bit-bang @ 0x2900-0x2903)	IEEE-488 PIA @ 0x2900-0x2903, bit-banged (PB3 strobe)
Host code	Driver wedge (DCL/ DFD.SPR)	Full BIOS replacement @ 0xDF00
Boot	QUICKPAC autoload from floppy	BOOTHd.COM + SYSGEN to winchester

Implications for the interposer project

1. **The ACT product validates the whole interposer concept on this exact machine** — a Z80-socket interposer was a shipping commercial Osborne 1 peripheral in 1982.
2. Its transport is the **same PIA bit-bang as Drive C** (`2900–2903`), so an ACT personality would reuse that machinery — but its command protocol is undocumented and would have to be reverse-engineered from `HARDBIOS.HEX` (`docs/harddisk-decision.md` §3).
3. The mass-storage decision is **made**: emulate the Adaptec ACB-4000 (the controller inside the Media Distributing **MD-10**), not the ACT. See `docs/harddisk-decision.md` for the reasoning and the plan.

Open questions

The ACT-specific questions are **closed or superseded** — the ACT is not being emulated (`docs/harddisk-decision.md` §3):

1. ~~Which ACT product exactly?~~ **Superseded.** `MOVCPM10/5/F.COM` are stock Digital Research `MOVCPM.COM` builds, not volume sizes.
2. ~~Command set at 0x2A00~~ — **resolved as not the disk.** The transport is the `2900–2903` PIA and its protocol is undocumented.
3. ~~The `OUT (C)` target port at 0xEF08~~ — **resolved**: the bank restore.
4. Whether any ACT hardware photos/schematics survive online — still unattested, and now only of archival interest.
5. ~~Re-derive the port claim~~ — **done** (above; `harddisk-decision.md` §3).
6. **The MD-10 / ACB-4000's** remaining questions — the DPB, the host-adapter bit map, the drive letters, "36 user areas" — are in `docs/harddisk-decision.md` §10.

The hard-disk decision: the ACB-4000

Status: Solved. As a decision: emulate the **Adaptec ACB-4000** SCSI-1 controller — the one inside the Mega-byte / Media Distributing **MD-10** — and **not** the ACT 1982 unit. One personality, built as a **faithful ACB-4000 SCSI target behind a configurable host-adapter shim**, so that the same firmware serves any Z80 machine with an ACB-4000-class driver and not only the Osborne 1. The build this authorises is #179; §10 carries what is still open.

This is the decision `docs/occ1-harddisk-analysis.md` was the groundwork for. It answers #8′s remaining checklist: the command set is documented (§4, from the vendor's own OEM manual), the recommendation is written (§2), and the ACT's port claim is read out of the ACT's own BIOS (§3). It is also the **formal plan** for the personality (§9).

The DPB is in the archive. `0x055E` is an **offset from the MD-10 BIOS's load address `0xD800`**, not a low-RAM address, so the per-drive type bytes sit at `0xDD5E`, the 24-byte descriptors at `0xDD6E` and the DPB at `0xDDB6` — all *static data inside the BIOS image*, read straight out of `HARD.COM` at file offsets `0x998`, `0x9A8` and `0x9F0` (§7.3; `docs/md10-system.md` §9). What is left is not the numbers but **who copies the static tables into a running system** (§10 item 1).

Sources: `research/harddisk/` (the ACT disk) and `research/adaptec/` (the MD-10 software) — both extracted and disassembled in-project; and the Adaptec ACB-4000 OEM manual, archived at `research/adaptec/acb4000/` (§4, and the archive note in §9.1). Serves #8; overlaps #20, #27, #84.

1. The decision, in one sentence

The interposer emulates an Adaptec ACB-4000 — a standard ANSI X3T9.2 SCSI-1 target — presented to the host through a small, configurable port block; the Osborne 1 configuration of that block is the one the MD-10 driver expects.

Two consequences worth stating up front, because they are the point:

- We are **not** emulating the MD-10′s host adapter as a one-off. We are emulating the **controller** the MD-10 was built on, and *describing* the host adapter as configuration. That is what makes the personality portable (§5).
- Nothing here needs the Osborne 1′s own floppy controller, IEEE port, or any other onboard device, so the personality **does not contend for a port** with Drive C, the printer or the RT-60A (§2). That is the opposite of the ACT route.

2. Why — the two candidates, measured

Both products are in the archive. Both were disassembled for this decision (`z80dasm`, scratch listings; the reusable commands are in the sources of `docs/occ1-harddisk-analysis.md`). The comparison is not close:

	ACT (1982)	**MD-10 / Adaptec ACB-4000**
Host interface	**bit-banged over the IEEE-488 PIA** `2900–2903`	its **own 4-register port block**, 4-aligned relocatable base (default `30h`)
Protocol documented?	**No** — must be reverse-engineered from `HARDBIOS.HEX`	**Yes** — SCSI-1 command set, handshake and a sample driver, all in the OEM manual
Host software	a **full BIOS replacement** (17-jump table; no floppy driver at all)	driver + BIOS/BDOS/CCP overlays, installed by `HARD.COM`
Geometry	5/10 MB, undocumented	**MODE SELECT descriptor** — block size, cylinders, heads, reduced-write-current, precomp, landing zone, step code
Port contention	shares **P3** with Drive C / printer / RT-60A	**none** — its own port block
Reverse engineering left	the entire protocol	*bounded*: which status bit is REQ / BSY / phase
Primary sources held	the driver disk only	the driver disk **and** schematic, photos, controller ROMs, two OEM manuals (§9.1)

Recommendation: emulate the ACB-4000. The deciding argument is not electrical convenience — it is that one route requires inventing a protocol and the other requires implementing a published one. The ACT's command set lives only in its own BIOS image; the ACB-4000's is a standard SCSI-1 CDB set that Adaptec printed, with a worked Z80 driver, a worked format sequence and a register map (§4).

3. The ACT — the ports it actually uses

##8 flagged the port claim in `docs/occ1-harddisk-analysis.md` — `0x2A00` (command/data) and `0x2C01` (status) — as suspect: those addresses "are the ACIA and video PIA, so something there is misread." The flag was right. Disassembling `HARDBIOS.HEX` (`z80dasm`, `org 0xDF00`, `0xDF00–0xEB00`) settles all three points:

1. **`0x2A00`/`0x2A01`/`0x2C01` are not the disk.** `0x2A00`/`0x2A01` are the console **ACIA** (`2A00` status read / control write, `2A01` RX/TX) — the console helpers `sub_e992h`/`sub_e99bh` read `2A00` and test **bit 1** = **TDRE** and **bit 0** = **RDRF** — and `0x2C01` is the **video PIA**. `docs/o1-memory-io-map.md` §3 already says so; the driver confirms it.

2. **The disk rides the IEEE-488 PIA at `2900–2903`.** It is the only device routine in the image, it is reached from the disk path — READ (`E32A`) ⟶ `sub_e448h` ⟶ `E475` ⟶ `sub_e520h` ⟶ `sub_e58fh` (`E58F`) — and it works by `set 3,(02902h)` / `res 3,(02902h)` as a **PB3 strobe** around `ld a,(02900h)` / `ld (02900h),a`, with `cpl` throughout for active-low. That is the *same transport Drive C uses.*
3. **The "indirect `OUT (C),A`, target TBD" is the bank restore.** `C` comes from `[0xEF08]`, which the two bank helpers set to `0` or `1`; `OUT (C),A` is simply re-selecting the bank on the way out. It is not a disk port.

The ACT's `MOVCPM10/5/F.COM` are **stock Digital Research `MOVCPM.COM`** builds (identical strings, a handful of patched bytes), *not* evidence of 5 MB / 10 MB volume sizes — so the ACT's version list is three builds of one utility, not three volume sizes.

Honest residue. The ACT board itself is unattested — no photograph or description has been found, and the `2900–2903` conclusion is read from the driver, not from a schematic. It does not change the decision: whether the ACT adapter *is* the PIA or merely *mirrors* PIA-style access, its protocol is undocumented and would have to be recovered byte by byte.

4. What the ACB-4000 actually is

From the OEM manual (`research/adaptec/acb4000/500504-00_ACB-4000_OEM_Manual_Mar84.pdf`):

- **A SCSI-1 target, not a "Z80 card."** It interfaces "to a host adapter according to the proposed ANSI X3T9.2 Standard (SCSI)", on a 50-pin ribbon, single-ended, all signals low-true. It drives **two ST-506/412** drives, has a **1 KiB dual-ported buffer**, and **variable sector lengths 256 B – 1 KiB** "programmed at format time."
- **The command set** (manual Table 6-1) is the standard SCSI-1 set — `00` Test Unit Ready, `01` Rezero Unit, `03` Request Sense, `04` Format Unit, `08` Read, `0A` Write, `0B` Seek, `0F` Translate, `13`/`14` Write/Read Buffer, `15` Mode Select, `1A` Mode Sense, `1B` Start/Stop Unit, `1C`/`1D` Receive/Send Diagnostic — with a Class 01 set (Read Capacity, …) as an option. Commands are CDBs; completion status is a byte (bit 1 = Check condition, bit 2 = Equal, bit 3 = Busy).
- **The reference host adapter** (Appendix A) is a three-register model: `BASE` = data, `BASE+1` = control (`SELECT 40h` asserts SEL), `BASE+2` = status (`REQ 80h`, `I/O 40h`, `MSG 20h`, `C/D 10h`, `BUSY 08h`), with the controller at **ID 01**. The manual ships a **sample Z80 driver** and a **sample format sequence** against exactly that model.
- **Geometry is host-supplied.** `MODE SELECT` carries a descriptor list — block size, cylinder count, head count, reduced-write-current cylinder, precomp, landing zone, stepping code. That is the same field set the MD-10's `DRIVETBL.DAT` and its drive-table editor carry ("Enter number of cylinders / heads / cyl. to start reduced write current / step pulse output rate code / landing zone position"), so the MD-10 is a *user of* the standard descriptor, not a private format.
- **The fill byte is a documented feature, and it is CP/M's.** `FORMAT UNIT` writes a default byte (`6Ch`) **or a user-selected one**, and the manual says so in as many words: *"This example will assume that the format is for a CP/M system that requires an **E5 (HEX)** data field."* A freshly-formatted volume is therefore one repeated byte — which is the whole basis of the sparse blob (§7).

- **The controller has its own brain.** The archive holds **six 8 KiB ROM images** plus an Intel-HEX build (`research/adaptec/acb4000/firmware/`), i.e. the ACB-4000 is a small computer running firmware, and its SCSI behaviour can be *read* rather than inferred. See §9.3 for how far we take that.

5. Portability — one personality, many Z80 machines

The ACB-4000 has **no "Z80 side."** Its host side is SCSI; what a Z80 machine has is a **host adapter** (a SCSI initiator), and *that* is the machine-specific part. The good news is that a host adapter is small and stereotyped — a data register, a control register with a `SEL` bit, and a status register with `REQ`/`I/O`/`MSG`/`C/D`/`BUSY` — and the OEM manual documents exactly that model as its reference (§4). So the personality is built in three layers, and **only the middle one is per-machine**:

Layer	What it is	Machine-specific?
SCSI target	ACB-4000 command set, phases, status/sense, two LUNs, `MODE SELECT` geometry, `FORMAT` fill byte	**No** — one implementation, to spec
Host-adapter shim	the Z80-side port block: base port, register roles, status bit map	**Yes** — a *profile*, not code
Backing store	the FAT32 blob, the mailbox, `MOUNT.COM`	No

Ship the shim with at least two profiles: **`md10`** (the Osborne 1 adapter — base `30h`, the observed bit map, §6) and **`reference`** (the OEM Appendix A model, §4). The first runs the period MD-10 software unchanged; the second runs any driver written against Adaptec's documented adapter. Add profiles as machines are found.

This is not a new idea — it is the original's. The MD-10′s host adapter was almost certainly a **Z80-socket device** itself (that is why it "does not tie up existing OSBORNE ports", why it can decode its own `IN`/`OUT` ports, and why the ad can promise it "will interface with any Z-80 computer … using CP/M, OASIS or other system"). Our interposer is doing what the original board did; we are simply making the host-adapter half a *description* instead of a fixed PCB, so that the SCSI half is written once. (Inference from the ad and the port map, not from a schematic of the MD-10 board.)

The Z80-socket form factor is the portability carrier: any machine with a socketed 40-pin Z80 can host the interposer, and the **bus side does not vary** — a Z80 socket is a Z80 socket, so the electrical interface is the same wherever the board fits. What varies between machines is **logical, not electrical**: (a) which port/register model *that machine's driver* expects, and (b) whether the machine already decodes those addresses — and the interposer overrides them either way. Both are configuration, so the shim is a *profile* (a table), not a board. The original MD-10 host adapter was a PCB, but nothing about the Z80 side of it needs to be.

6. The Osborne 1 shape of it (the MD-10)

From the MD-10 software (`research/adaptec/`, disassembled):

- **Port base is a parameter.** `HARDBIOS.SPR` is built for base **`30h`**; the shipped `HD00BIOS.SPR`/`HD04BIOS.SPR` are the *same code* at base **`00h`** and **`04h`** — that is the "must be on a 4 port boundary" prompt — and `PORTCHNG.COM` is a **file patcher** (it contains no `IN`/`OUT` of its own) that rewrites the base. So the profile's base port is exactly the thing `PORTCHNG` edits.
- **Registers:** `base+0` data (also the `OUTI`/`INI` block port), `base+1` control, `base+2` status read / control write, `base+3` control. Status bits the driver polls: `01h`, `02h`, `20h`, `40h`. The probe, the transaction and the bit meanings are read out in `docs/acb4000-host-adapter.md` §1 — including that `base+2` reading `04h` is "exactly one controller on-line", and that `30h–33h` **alias onto the on-board 2-bit I/O decode** (so the adapter must own the bus for those ports; §2).
- **Host software:** `HARD.COM` installs; `HARDBIOS.SPR`, `HARDBDOS.SPR` and `HARDCCP.SPR` overlay BIOS/BDOS/CCP (the same *shape* as the ACT, but over a documented controller).
- **Drive letters — the boot letters.** The MD-10 advertisement is explicit: *"With a single hard disk installation, the MD-10 10 subsystem becomes units A and B with the standard drives being designated E and F. If a second MD-10 is installed later it becomes units C and D."* The hard disk takes `A:`/`B:` because the hard disk is the **boot** device — CP/M boots from `A:`. The two letters are the ACB-4000's **two LUNs**.
- **User areas:** the ad claims *"36 different user areas per MD-10"* — beyond CP/M 2.2's 32. Whether that is the driver extending user areas or just loose copy is a sub-issue (§9.4).
- **Memory overhead:** the ad claims *"less than 2K"* of TPA for the driver.

7. What we can offer — sizes, the CP/M 2.2 ceiling, and the sparse blob

"What size disks can we offer?" Three limits stack, and the smallest wins:

Limit	Where it comes from	Value
The controller	ACB-4000: two ST-506 drives, sector 256 B – 1 KiB, **geometry is host-supplied**	two LUNs, any geometry the drive table defines
The driver's DPB	MD-10: `DRIVETBL.DAT` + the drive-table editor set cylinders / heads / RWC / step / landing zone	per-logical-drive size is whatever the DPB says
CP/M 2.2	`DSM` is 16 bits; drives `A:`–`P:`; max file 8 MB	≈1 GiB per drive at 16 KiB blocks; 8 MB per file

The advertised envelope is **10 MB (MD-10), 20 MB (MD-20), "20 and 44 MB also available."** Because the controller auto-configures to the host-supplied geometry and we back it with a blob, **our practical limit is the driver's DPB, not our storage** — we can offer any geometry `DRIVETBL.DAT` can describe. The drive table's fields are read (§7.2) and so is the DPB (§7.3), so the geometry side of that is settled and #179 has a default; what is not chased is which copy of the tables a boot actually uses (§10 item 1).

"How does it handle CP/M 2.2's limits on the Osborne?"

- **16 logical drives** is CP/M's limit, not the device's — the same conclusion `docs/drive-c-protocol.md` §8.4 reached for Drive C.
- **8 MB per file** is CP/M 2.2's extent limit; the ad's "read or write a 64K file in less than four seconds" sits comfortably inside it.
- The O1's **stock BIOS is fixed-size with floppy-sized DPBs**, so the MD-10 **replaces BIOS + BDOS + CCP** (`HARDBIOS/HARDBDOS/HARDCCP.SPR`) rather than wedging into the existing jump table. That is the mechanism by which a bigger DPB and more drives become possible at all.
- TPA overhead: claimed < 2 KiB. User areas: the ad claims **36** (> CP/M's 32).

The sparse blob. The ACB-4000's `FORMAT` writes a fill byte across the whole drive — default `6Ch`, and `E5h` "for a CP/M system" (§4) — so a freshly formatted, empty volume is *one repeated byte*, and a used volume is that byte with islands of data. Two encodings, both self-describing:

- **Blank volume:** `(fill_byte, geometry)` — effectively zero bytes.
- **Truncated:** the non-fill extent, plus the fill byte and the full geometry, reconstituted on load.

This is the same "the size *is* the geometry" trick as the Drive C blob (`docs/drive-c-design.md` §4.2), with one addition the user's original sketch did not need: **record the fill byte rather than assuming `0xE5`.** The controller's default is `6Ch` and CP/M's is `E5h`, so the fill is a property of the *volume*, not of CP/M.

One caution. CP/M writes `E5` to the **directory** eagerly but not the whole data area (`docs/drive-c-protocol.md` §8.5, measured on the Drive C install), whereas the ACB-4000's `FORMAT` is what makes the *whole* volume uniform. So a volume is only "sparse-friendly" if it was formatted by the controller — which it will be — and the trailing-fill length should be **measured on a formatted volume** before the tooling trusts it.

7.1 What #178 found — directories, not loose files

Reading the MD-10 BIOS overlay `HARDBIOS.SPR` (the file maps to address `0xE000 + offset`; the BIOS jump table is at `0xE100`, and its entries are relative to that base, so `SELDSK` at `0xE1D5`):

- **It is a CP/M 2.2 filesystem.** `SELDSK` returns a **DPH**, so the volume is an ordinary CP/M logical drive with a **directory** — `DIR`, `PIP`, `ERA`, `STAT` work, and files are normal CP/M files. It is *not* a raw 10 MB blob of loose files.
- **Up to 16 logical drives, not two.** `SELDSK` opens with `LD A,C` / `CP 10h` (`0xE1D5–0xE1D7`) — the driver can present all of CP/M's `A:–P:`. The advertisement's "units A and B" is the shipped *default*, not a limit.
- **The geometry is data-driven, which is why there is no literal DPB.** `SELDSK` indexes a **per-drive type byte** at `0x055E` (one per drive; `0FFh` = drive absent), then a table of **24-byte per-type descriptors at `0x056E`**, selecting entry `type × 24`; the returned DPH sits at entry `+8`. The per-drive track limit lives at `0x016F`/`0x0170`. So the DPB is *built* from the drive table — the same reason `DRIVETBL.DAT` is editable.
- **Management is standard CP/M plus the vendor's own tools.** `MD10.COM` / `MD20.COM` is the drive-table editor *and* `PREP` — format with a label and a password, two-pass format then verify.

PORTCHNG.COM relocates the port base. The companion disk carries the transfer/backup set (AREN, DIRA/DIRS, GET/PUT, ERQ, FSET, NS, OSD, POP, SUBMIT, WSPATCH.*).

- **The controller driver is a separate overlay, and it is port-parameterised.** CNTLRCOD.OVL — and CNT00COD.OVL / CNT04COD.OVL, the *same code* at port bases 30h / 00h / 04h — loads at **0x8000**. The SCSI handshake is right there: poll base+2, write the command byte held at 0x8023 to base+0, then block-transfer with OUTI/INI. And the ACB-4000's default format fill byte (6Ch, §4) sits at 0x8024 as a lone data byte — an independent confirmation of the controller, found in the driver rather than the manual.
- **DRIVETBL.DAT is a list of drive-model entries.** This is the table MD10.COM edits and SELDSK's 24-byte descriptors are built from. Its shape is not what its own *directory entry* says: the entry claims 128 bytes while the file owns nine allocation blocks and holds 16 512 bytes of live data. **§7.2 has the structure and the byte→field mapping.**
- **The DPB is static data in the BIOS image**, not a table built in low RAM. The per-drive type bytes, the 24-byte descriptors and the DPB itself are read out of HARD.COM at 0xDD5E, 0xDD6E and 0xDDB6 (§7.3). What is not yet chased is which copy a boot uses — the static one, or the drive's own copy of the geometry and type bytes at LBA 4 (docs/md10-system.md §3, §9.1).

7.2 The drive table — structure, and the byte→field mapping (#178)

Solved, by reading the shipped file and then running the editor that edits it.

Two things about the file's shape hide the mapping, and both are read from the file rather than assumed:

- **DRIVETBL.DAT is not one 128-byte record.** Its directory entry says `rc = 1` (128 bytes) and even sits at extent 1, but the file owns nine allocation blocks and holds **16 512 bytes of live data**. A `cpmfs.py get` therefore returns the header and five name records and nothing else; the rest is reachable only by walking the block list (tools/research/drivetbl.py).
- **A record is not a 7-byte name plus four parameter bytes.** The name field is **ten** bytes, and the four bytes are not the numeric fields at all — the geometry lives somewhere else entirely.

The file

Offset	Length	Holds
0x0000	13	**Header.** +0..1 = 13 + 11×N, the end offset of the name table (LE); +2 = the next free description-record index (0x81 = 129 here); +3 = 0x0D = 13 — the header length, i.e. where the records start; +4..12 = 0
0x000D	11×N	**126 name records** of 11 bytes: a NUL-terminated name of up to 10 characters, then **one byte — the index of that drive's description record**
13+11N	9	**Sentinel** 00 "RDBAK" 00 00 00. Its leading 0x00 is what stops the driver's name search
0x0800	128×113	**113 description records**, one per index 16…128, at file offset 128 × index — so records 11–15 of the file are a gap between the two areas
desc. record	+0..8	**the nine geometry bytes**
	+9..127	the free-text description, NUL-terminated, with the stale tails of longer earlier strings left behind

The nine geometry bytes — the mapping

They are **not packed**. They are the ACB-4000's own MODE SELECT drive-parameter block, field for field, in the manual's order (parameter-list offsets 0D–15):

Description record	Manual (`MODE SELECT`)	Field	Stored as
`+0..1`	`0D–0E`	Cylinder count	16-bit, **little-endian**
`+2`	`0F`	Head count	8-bit
`+3..4`	`10–11`	Reduced-write-current cylinder	16-bit, LE
`+5..6`	`12–13`	Write-precompensation cylinder	16-bit, LE
`+7`	`14`	Landing zone	8-bit (signed in the editor: `Lz=+0`)
`+8`	`15`	Step pulse output rate code	8-bit

The manual's block is big-endian and the file is little-endian, so the driver byte-swaps when it builds the descriptor. **That byte order is the whole of it** — nothing here is packed; five fields in eight bytes was never the shape of the thing.

Two of the fields are outside the editor: it prompts for cylinders, heads, reduced-write-current, landing zone and step code (measured, below) and neither displays nor writes `+5..6`. The manual puts write precompensation there and every entry in the shipped table has it zero, so **precomp is inferred**, not measured.

The name record

An 11-byte record is a NUL-terminated name of up to ten characters and then the description index. The driver's name search compares at most ten bytes and stops at the typed name's terminator (`LD B,9` / `DJNZ`, `0x0C04–0x0C12`), so **the bytes after the name's NUL are never read by the driver**. In the shipped table they are not consistently name text (`AT3020` is followed by `"15"`, the Seagate entries by `"62"`, `D-720` by three unprintable bytes), so their meaning is unresolved and, for #179, irrelevant.

Bit 7 of a name's **first** byte is a flag — the search masks it off with `AND 7Fh`. In this table it is set on 26 entries, and every one of them is an alias: all 26 share their description record with another name (the editor's "Add <S>ynonym"), and no unflagged entry shares one.

How we know

Measured, in MAME, by running the editor itself. `MD10.COM` will not start without an ACB-4000 answering on `30h–33h`, so **three** 3-byte patches were applied (`tools/research/md10patch.py`): the overlay's init call at `0x0219`, which is what exits the program when nothing answers; the probe at `0x15AF`; and the branch at `0x0EE8`, which takes the loaded table to the editor's menu instead of its "name not found" path. Nothing else in the program is touched — the geometry code, the editor and the file I/O are the vendor's own bytes. With that, the editor's own `<L>ist parameters` prints, for `CM5205`:

```
CM5205: Computer Memories, Inc. model CM5205
  Cyl=256, Head=2, Reduc=256, Lz=+0, Step=1, Cap=4608K
```

and the description record at index 16 is `00 01 | 02 | 00 01 | 00 00 | 00 | 01` — 256, 2, 256, 0, 0, 1. Field for field, including the little-endian pair. The same run fixed the rest:

- the editor's prompts run **cylinders, heads, reduced-write-current, landing zone, step code** — ascending record offsets 0, 2, 3, 7, 8;
- pointed at the 13-byte *header* instead of a record, the same display reads the header's bytes at those offsets — `Cyl=1399, Head=129, Reduc=13, Lz=+0, Step=0` is `77 05` / `81` / `0d 00` / `00` / `00` — an independent check of the offsets from the measured display;
- the loader reads 11 CP/M records (1408 bytes) into `0x3001` and sets the current-record offset at `0x3004` to **13** — the header length.

Read from the primary source: the shipped `DRIVETBL.DAT` itself, dumped by `tools/research/drivetbl.py`. **Read from the driver**: the write side uses the same five offsets (`LD (2719h),HL` … `LD (2721h),A`). **Cross-checked against the manual and against the real drives**: the step codes are the manual's table (`0` = 3 ms unbuffered, ST-506; `1` = 28 µs buffered, ST-412; `2` = 12 µs buffered, most others) and the shipped entries obey it exactly — `ST506` ⟶ 0, `ST412` ⟶ 1, everything else ⟶ 2; and the geometries are the drives' own: `ST506` 153/4, `ST412` 306/4, `ST419` 306/6, `SA604` 416/4, `XT-1065` 918/7, `XT-1140` 918/15, `XT-2085` 1224/7.

Not obtained: a write-back diff. In the emulated run the editor took an edit, prompted through all five fields and returned to its menu without rewriting `DRIVETBL.DAT` — the only bytes that changed in the image were our own patch — so the write side is read from the code, not measured.

What this gives #179

A default geometry, and the table to pick it from. The editor's own template for a new entry is the name `PYXIS13`, and that name is in the table — the vendor's default drive. A selection from it:

Entry	Cyl	Heads	RWC	LZ	Step	Cap	Description
PYXIS13	**321**	**4**	**132**	**0**	**2**	11 556K	Rodime PLC model RO 202 / Ampex model Pyxis 13
ST506	153	4	128	0	0	5 508K	Seagate model ST506 — the manual's reference drive
ST412	306	4	306	0	1	11 016K	Seagate model ST412
RO202E	640	4	0	0	2	23 040K	Rodime PLC model RO 202E
PYXIS27	321	8	132	0	2	23 112K	Rodime PLC model RO 204 / Ampex model Pyxis 27

Recommendation: PYXIS13. It is what MD10.COM itself offers a new drive, it lands in the MD-10′s advertised 10 MB class, and period software and the driver accept it by construction. ST506 is worth keeping as a second profile: it is the manual's worked example and the only step-code-0 case, so it exercises a different code path. Cap is the editor's own formula, cylinders × heads × 18 × 512 B, which reproduces its Cap=4608K for CM5205 exactly; the personality need not agree with it, but #179′s MODE SENSE must report the geometry the table holds.

The whole table — 126 entries over 113 description records — is in research/adaptec/MD Utilities Adaptec 4000 Controller Ver. 1.93 Osborne - MD-10 MD-20.img; tools/research/drivetbl.py prints it. The 2.01 disk carries the same table byte-for-byte.

A second, independent list. `DRIVETBL.DAT` is Osborne's/MD-10's list. The period list archived at `research/adaptec/compatible-drives-1991-usenet.txt` (Robert Haar, `comp.sys.3b1`, Feb 1991, reposting Lenny Tropiano's 1989 table out of the Western Digital support BBS) is the *drive manufacturers'* one, and it is broader: Atasi, ALPS, Ampex, CMI, Disctron, Evotek, Fujitsu, IMI, Maxtor, Microscience, Miniscribe, Mitsubishi, NEC, Newbury, Priam, Rodime, Seagate, Sony, Tandon, Teac, Texas Instruments, Toshiba, Tulin, Vertex and Western Digital. It also gives each drive's **sectors/track** — 17 for almost every entry — which `DRIVETBL.DAT` does not carry at all (its capacity formula assumes 18), and it numbers `PYXIS-13` at **320** cylinders where `DRIVETBL.DAT` says **321**. Neither is wrong: one is the drive's own number, the other is what the tools carry. Nothing is reconciled here; both are recorded, because the personality (#179) may offer any geometry the driver's DPB can describe, not only the 126 in the shipped table.

Making a drive image. `tools/research/mkdrive.py` builds a blank-but-formatted ST-506/MFM image for any geometry, byte-wise, without the 1984 tools:

```
tools/research/mkdrive.py --out /tmp/md10/hd.chd                # PYXIS13
tools/research/mkdrive.py --out /tmp/md10/st412.chd --chs 306,4,17
```

It fills every sector's data field with **`6C`**, which is what the ACB-4000's FORMAT writes by default — "completely written with 6C hex unless otherwise specified in the format command" (`research/adaptec/acb4000/400003-00A_ACB-4000A_Users_Manual_Oct85.ocr.txt`, §4.4.5 and the FORMAT command reference) — and passes the geometry to `chdman`, whose `GDDD` metadata is the same record the controller keeps in the disk's ID fields. `PREP.COM` asks for a *unique* fill byte and sends `D9` in its FORMAT CDB, so `--fill d9` exists too. It is needed because `PREP.COM` cannot finish its own format under MAME — the verify pass that follows it (`2F`) has no handler (`docs/osborne1nx-machine.md` §5).

The drive's *own* structure — what `PREP.COM` writes once it has finished a format — is read: it is the four-block header at LBA 0–3 read out of the program, with the disk label and its 28-byte partition records at LBA 0. `docs/osborne1nx-machine.md` §6 has the field map and the recipe.

Running the editor

Four things were added to `tools/research/` for this, each with a header saying what it was for: `o1raw2imd.py` (wrap an archive `.img` for MAME, which will not open a bare raw sector image), `o1keys.lua` + `o1keys.sh` (a scripted-keystroke harness, with a `peek` directive — that is what made the in-memory table checkable), `md10patch.py` (patch a file inside the image, by name; the three patches above) and `drivetbl.py` (dump the table).

7.3 The DPB — read out of `HARD.COM`, with its values and its mechanism (#178)

The DPB is in the archive. #178's acceptance is "the DPB for each logical drive is written down with its source offset", and both halves are now read: the **values**, out of the file, and the **mechanism**, out of the code.

The audit that produced a negative result here read `0x055E` as a *low-RAM* address. It is not one. It is an **offset from the BIOS's load address**, and the MD-10 BIOS loads at `0xD800` (`docs/md10-system.md` §4), so:

What	Address	HARD.COM file offset	Size
per-drive type byte table	0xDD5E	0x0998	16 bytes
24-byte per-type descriptors	0xDD6E	0x09A8	24 × type
the DPB	0xDDB6	0x09F0	15 bytes

All three are **static data inside the BIOS image** — the address less the loader's 0x0100 origin — so they are read out of the file rather than watched at run time. docs/md10-system.md §9 read them out and cross-checked them against the driver's code; the values below are that reading, and this section does not restate its §9.1.

The DPB itself — 15 bytes at HARD.COM file 0x09F0:

```
file 0x09F0   9C F0 | 05 | 1F | 01 | 83 07 | FF 03 | FF | 00 | 00 00 | 00 00
              SPT   | BSH| BLM| EXM| DSM   | DRM   | AL0| AL1| CKS   | OFF
              F09Ch | 5  | 31 | 1  | 1923  | 1023  | FF | 00 | 0     | 0
```

DPB+3 is 31 — the BLM read independently out of the driver's *code* ("the driver reads DPB+3 as BLM"). Two readings agreeing is what makes 0xDDB6 the DPB rather than a lookalike.

BLM = 31 is 4096-byte blocks, so DSM = 1923 is 7.88 MB per logical drive; DRM = 1023 is 1024 directory entries; AL0 = FFh is the eight directory blocks; CKS = 0 is a fixed disk, which is right. **SPT reads F09Ch = 61 596, and that is not a sector count: it is the partition measured in 128-byte records** — 61596 × 128 = 7 884 288 = 15399 × 512, exactly one 15399-block partition. So the DPB does not merely *describe* a partition size, it **is** the partition size, and it is the same 15399 blocks PREP.COM writes when it is given a PYXIS27 (docs/md10-system.md §3).

One DPB, shared by every hard-disk drive. The three descriptors carry DPB 0xDDB6, DIRBUF 0xEE80 and CSV 0 in common; only the allocation vector differs, at 0xDDC5/0xDEB6/0xDFA7 — exactly 0xF1 = 241 bytes apart, which is the ALV size for DSM = 1923. Two consequences, both load-bearing for #179: every hard-disk partition is the same size in CP/M's view, and the only layout in which the drive and the driver agree is **three 15399-block partitions on a PYXIS27**. docs/md10-system.md §9.1 has the descriptors and the arithmetic.

The chain, with source offsets

All offsets are file offsets into HARDBIOS.SPR, which maps to 0xE000 + offset. It is a **relocatable** image: the BIOS jump table is at file offset 0x100 (0xE100) and its operands — and the routine's own internal JP/CALL targets — are *offsets from the table*, not addresses. That is why the table below names 0x055E/0x056E: they are the operands as written, and they resolve to 0xDD5E/0xDD6E once the BIOS is installed at 0xD800. Read the image linearly and the disassembler drifts through the data and invents instructions; tools/research/hardbios.py resolves them the way the loader does.

Step	Offset	What it does
SELDSK entry	0x11B	jump table: operand 0x00D5 ⟶ target 0x1D5
drive range	0x1D5	LD A,C / CP 010h / RET NC — drives 0–15, else NULL
type byte	0x1DC–0x1E7	saves the drive at 0x0180, indexes 0x055E + drive; 0FFh = absent
descriptor	0x1F6–0x203	type * 8, then ×3 more, then + 0x056E ⟶ 0x056E + type*24
the DPH	0x204–0x209	+ 8, saved to 0x0555, returned in HL
the DPB pointer	0x393–0x39C	in READ/WRITE: HL = (0x0555) — the saved DPH — +10, dereference ⟶ **the DPB**
block size	0x39D–0x3A2	+3 into the DPB = **BLM**, +1 = the deblocking count
sector size	0x3DA–0x3E4	SRL H / RR L **twice** — the 128-byte CP/M record number shifts right two, so the physical sector is **512 bytes**

Three things follow, and they are the substance of the answer:

1. **There is a DPB per logical drive, and it is an ordinary CP/M 2.2 DPB.** The driver reaches it at DPH+10 — the DPH's DPB field in the canonical layout (XLT 0–1, scratch 2–7, DIRBUF 8–9, **DPB 10–11**, CSV 12–13, ALV 14–15) — and then reads DPB+3 as BLM. So the descriptor points at a 15-byte DPB whose fields are the usual SPT/BSH/BLM/EXM/DSM/DRM/AL0/AL1/CKS/OFF.
2. **SPT is counted in 128-byte records, four to a physical sector.** The record⟶sector conversion is a two-bit shift (0x3DA), so a 17-sector track is SPT = 68. Whoever turns §7.2′s geometry into a DPB needs that, and it is read from the code rather than assumed.
3. **A descriptor entry is 24 bytes with the DPH at +8**, so it carries eight bytes *before* the DPH. What those eight hold is not determined — nothing in the archive reads them.

How the tables were found, and why the first search missed them

tools/research/hardbios.py audit searched for a *writer* of 0x055E/0x056E, treating them as absolute low-RAM addresses, and swept 0x0500–0x0700:

```
=== writers of 0x0500-0x0700 ===
  HARDBIOS.SPR       0555@0x209 0557@0x18D 0557@0x217 0558@0x1E9
  ...
  0x055E / 0x056E writers: NONE
```

Three things follow, and together they are why the tables read as "missing":

- **The sweep found the relocatable *operands*, not the data.** `LD BC,055E` / `LD DE,056E` inside the overlay are reported (`LD DE,056E` at `hardbios.spr` offset `0x200`); the tables they index live at `0xDD5E`/`0xDD6E`, outside the searched window. A store to a BIOS-relative address is not a store to `0x0500–0x0700`.
- **The tables are not in the `.SPR` files at all.** A byte search for the 16-byte type run, the `01 14 00 00 27 3C` descriptor prefix and the ten-byte DPB prefix finds all three in `HARD.COM` and in **none** of `HARDBIOS.SPR`, `HARDBDOS.SPR` or `HARDCCP.SPR`. The `.SPR` is a *different, larger* build — its body is `0x880` bytes against the embedded copy's `0x5C5` — the same BIOS at the same entry points but not the same image. The driver's fallback table therefore lives only in the loader that installs it, which is why #178's audit of the `.SPR` files found no writer.
- **The writes the audit does find are the BIOS's own variables** — `0x0555` (the saved DPH), `0x0557` (a flag), `0x0558` (the saved type), i.e. `0xDD55`, `0xDD57`, `0xDD58`. The two tables `SELDSK` *reads* are read-only.

Two further checks close the alternatives:

- **Not data carried by another overlay.** If an overlay were loaded so that its bytes landed on the tables, the table would be there as *data*. It is not: those offsets in `HARDBDOS.SPR` and `HARDCCP.SPR` are code, and neither file contains a per-drive type run (`00 01 02 … FF`) or a 24-byte record table.
- **Not in a system image — and it does not need to be.** The MD utilities disk is a **data** disk: its system tracks (tracks 0–2, 7680 bytes) are **all `0xE5`**, and no part of `HARDBIOS.SPR` appears in them. The tables arrive with the BIOS instead, inside `HARD.COM`, which carries a relocated copy of the BIOS and copies it to `0xD800` (`docs/md10-system.md` §4–§6) — so they are *installed*, not loaded from a system image.

The geometry the DPB is built from also lives **on the drive**, written by the last `FORMAT`; the next subsection is the manual's account of that, and it is what the personality has to reproduce.

Where the drive parameters live, and how they get written

The manual answers the half the archive does not cover — where the parameters go on the drive, and how they get there. Three commands, read from the Mar84 OEM manual §6.3.1:

- **`MODE SELECT` (15H)** — "is used in ACB controllers to specify FORMAT parameters and should always precede the FORMAT command." Byte 4 of the CDB is the byte count: 12 (`0Ch`) minimum, **22 (16h) when the drive parameter list is included**. The list is the parameter list (4 bytes, format code `01`) + the extent descriptor list (8) + the drive parameter list.
- **`FORMAT UNIT` (04H)** — "will write from index to index all ID and data fields with a block size as specified by an immediately previous MODE SELECT (15H) command."

- **`MODE SENSE` (1AH)** — "used to interrogate the ACB-4000 device parameter table to determine the specific characteristics of any disk drive currently attached. **The attached drive must have been formatted by an ACB-4000 for this to be a legal command.**"

And **where** on the drive — from the ACB-4000A user manual §2.3.1, which is an image-only scan (OCR committed beside it per `research/README.md`):

> During FORMAT the disk parameters provided by the MODE SELECT command will be encoded onto the disk. **These parameters are written to the ID field of cylinder 6.** This information is then read to the controller during the power-up initialization process. Access to these parameters is also available using the MODE SENSE command.
>
> This scheme obsoletes the need for the host software to initialize the controller on power up.

§3.7 adds that at power-up the controller "will seek the drive to the last cylinder and read the **largest block address** present. The parameter information and largest block address are saved on the … Controller" — so the **capacity** is read back at power-up as well as the geometry.

The parameters **persist on the drive**, and the manual says so three ways:

> If these parameters are not supplied the format operation will use **previously supplied values if available** or the default values given below.

> …the controller being unable to read the drive information **from a drive already formatted** *(the `1C` blown-format error)*

> Note that it is possible to specify only the block size in a MODE SELECT. In this case **the previous drive parameters are preserved**. *(May83 preliminary manual)*

So the geometry is written **onto the drive** by the last `FORMAT` and read back by `MODE SENSE` — precisely the factory-assembled case, where an ACB and its drive were paired and formatted before an end user saw either, so the drive already carries its parameters and a later `FORMAT` without a `MODE SELECT` keeps them. That is what makes `MODE SENSE` (1AH) and `READ CAPACITY` (25H) a **correctness requirement** for the personality (`docs/acb4000-firmware.md`), not a convenience.

The defaults, for a drive formatted with no parameter list at all — which is what an unparameterised volume reports:

Field	Default	Range
Cylinder count	**306**	1–2048
Data head count	**2**	1–16
Reduced-write-current cylinder	**150**	0–2047
Write precompensation cylinder	*ignored on the ACB-4000* (tied to RWC)	—
Landing zone	**0** (innermost)	7 bits + direction bit
Step rate code	`00` 3.0 ms unbuffered (ST-506); `01` 28 µs buffered (ST-412); `02` 12 µs buffered	—

What the manual pins, and how it relates to the DPB

The manual is about the ACB-4000, not about CP/M, so it carries no DPB. It does carry two of the DPB's inputs, and both bear on the one the driver holds:

- **Sectoring.** `FORMAT` writes "**the normal 32 sectors**" of 256 bytes per track — **33** when the interleave is ≥ 2 — and the block size is 256, 512 or 1024 B. 32 × 256 = **8192 bytes per track**; the driver's record⟶sector conversion is a two-bit shift (`record >> 2`), i.e. 512-byte physical sectors, 16 of them. Either way a track is 8 KiB, which is **64 × 128-byte CP/M records** — the arithmetic behind an `SPT` field, even though the shipped DPB's `SPT` holds the *partition* size instead (above). Read from the manual *and* the code, not assumed.
- **Capacity.** `READ CAPACITY` (25H) returns "the address of the last block on the unit" plus the block size, in an 8-byte data field. That is a `DSM` input the driver can take straight from the controller, and the first place to look before assuming `DSM` is computed from cylinders × heads.

Which of the two the driver actually uses at boot — the controller's report, or the static fallback in `HARD.COM` — is the open question, and it is the same one as "which copy of the tables" (§10 item 1). `docs/md10-system.md` §3 and §9.1 have the evidence on both sides.

What is left, and how to watch it

The tables and the DPB are read, so what remains is not "find the table". It is two narrower questions, and both are reachable on the running system:

1. **Which copy of the tables a boot uses.** The BIOS image carries them (`HARD.COM` ⟶ `0xD800`), and the drive carries its own copy of the geometry and type bytes at LBA 4 (`docs/md10-system.md` §3); the two disagree in size (`docs/md10-system.md` §9.1). Who copies one into the other is not chased.
2. **The peek that was going to answer it.** #191's `peek 0x055E/0x056E` is now *reachable* — the MD-10 system boots to a prompt with three partitions (`docs/md10-system.md` §7) — so the runtime tables can be read against a running system rather than inferred. The address to peek is `0xDD5E/0xDD6E`, not `0x055E/0x056E`.

Both routes the search left open are now closed, and neither was closed by giving up:

1. ~~A stub ACB-4000 in MAME.~~ **Tried, and blocked** — MAME's `osborne1` decodes only two I/O address bits, so `30h–33h` alias onto the ROM-bank ports and a Lua stub cannot reach the adapter; it needed a MAME driver change (`docs/acb4000-host-adapter.md` §2). The precedent was `tools/research/drive-c-box.lua`, a Lua model of the Drive C box, with which "the 1983 loader completes and PIP writes to `C:`" — the same move, for the same reason. **The driver change was built: `#191`'s `osborne1nx` machine**, which widens the decode to eight bits while keeping the mainboard's 2-bit alias, and the drive-table editor runs on it.
2. ~~A bootable MD-10 system disk.~~ **Produced, from the archive's own tools** — `PREP.COM` option 3 *configures an operating system onto the drive*, and the system it writes boots to a prompt (`docs/md10-system.md` §1, §7). The *on-disk* half of it — the four-block header at LBA 0–3, the label at LBA 0 and the 28-byte partition records it carries — is read out of `PREP.COM` and reproduced on the emulated drive (`docs/osborne1nx-machine.md` §6).

The host-adapter register protocol is read out in `docs/acb4000-host-adapter.md`, and the status-bit map this document's §10 item 2 asked for is answered there — the driver polls `01h`, `02h`, `20h`, `40h`.

`research/harddisk/OCC1_HARDDISK.IMD` is a *different* controller — Australian Computer & Telecommunications, reached through its own BIOS's controller-address prompt rather than the MD-10's fixed `30h–33h` window (§3, and `docs/occ1-harddisk-analysis.md` §3) — i.e. the RT-60A situation again: recorded as incompatible in `docs/occ1-harddisk-analysis.md` §3, moved past, and not part of this question.

8. Coexistence — the A/B→E/F remap and the CoPower-88

The MD-10 moves the O1's floppies from `A:/B:` to `E:/F:` (§6). Does that hurt the CoPower-88? The honest answer is **no show-stopper, but two real integration questions** — and neither is hardware.

1. **No port conflict.** The CoPower-88 host driver talks over its own port pair (`0x7E/0x7F` on Kaypro; the Osborne decode is still open, `#5`), while the MD-10 uses `30h–33h`. Different ports, no arbitration (`docs/copower88-protocol.md`).
2. **Two things patching the BIOS.** The MD-10 *replaces* the BIOS; the CoPower-88 host driver *installs a BIOS wedge* and asks for a drive letter `A–P` at install time. Anything that patches the jump table has to be installed against the BIOS that is actually resident, so **order matters**: the CoPower-88 wedge must go on after the MD-10 BIOS (or be made aware of it). This is the same class of problem the boot disk (#28) already has to solve for its other drivers.
3. **Drive letters are configuration, not fate.** The CoPower-88 loader *asks* which host drive to use, and the Drive C precedent shows a device's letter is renamable (`docs/drive-c-protocol.md` §8.4) — so the `A:/B:`⟶`E:/F:` move is survivable, at the cost of redoing the 8088-side DOS environment.
4. **It is not MD-10-specific.** Any driver that makes the hard disk the boot drive renames the floppies, because CP/M boots from `A:`. The MD-10 is not unusually hostile here.
5. **And we own a choice.** Because we control both the personality and the boot disk, we *can* keep the floppies at `A:/B:` and put the hard disk at `C:/D:` **if** the MD-10 driver's letter assignment is

reconfigurable. That trades vendor-faithful behaviour for CoPower-88 friendliness and should be a deliberate decision, not a default.

9. The plan

9.1 Archive the primary sources — first, so citations resolve

The ACB-4000 is now the chosen controller, so its primary sources belong in the in-repo archive, with provenance rows in `research/README.md` (AGENTS.md). Under `research/adaptec/acb4000/`:

Path	What it is
`500504-00_ACB-4000_OEM_Manual_Mar84.pdf`	the OEM manual — command set, host-adapter model, sample driver, sample format
`400003-00A_ACB-4000A_Users_Manual_Oct85.pdf`	the ACB-4000A user manual
`ACB_4000_and_5000_Disk_Controllers_OEM_Manual_Preliminary_May85.pdf`	the preliminary OEM manual
`ACB4000A_Schematic_ARD_2025.pdf`	the board schematic
`firmware/`	six 8 KiB controller ROM images + an Intel-HEX build
`pictures/`	seven board photographs

Source: bitsavers, `https://www.bitsavers.org/pdf/adaptec/ACB-4000/`.

9.2 The personality

Build the three layers of §5 in this order, each testable on its own:

1. **SCSI target** — the ACB-4000 command set, phases, `MODE SELECT` geometry, `FORMAT` fill byte, two LUNs. Written to the manual, machine-independent.
2. **Host-adapter shim** — ship the `md10` profile (base `30h`, the observed bit map) and the `reference` profile (OEM Appendix A). Profile = data, not code.
3. **Backing store** — the sparse blob (§7) behind the FAT32 library, and the mailbox/`MOUNT.COM` front end (§9.4).
4. **Bring-up** — run the *period* MD-10 software (`HARD.COM`, the `.SPR` overlays) against the personality, exactly as #84′s acceptance test runs the period Drive C software. That is the known-good test suite, and the reason a "we hold the drivers" personality is worth building at all.

9.3 How faithful to the controller?

We hold the controller's own ROMs (§4), so a bit-exact ACB-4000 is *possible* — run the firmware in a small emulator. **Recommendation: don't, for v1.** Implement the documented behaviour and keep the ROMs as the **oracle** to test against. Running the real firmware drags in the drive side (ST-506, the ACB-4000 does the sectoring and ECC) for no behavioural gain over the manual. Revisit only if a driver turns out to depend on undocumented controller behaviour.

9.4 The sub-issues

#	Sub-issue	Area	Depends on	Filed
1	**Read the MD-10 DPB and `DRIVETBL.DAT`** — pin the volume sizes and the drive-table fields	`research`	—	#178
2	**Archive the ACB-4000 primary sources** (§9.1) — **delivered with this decision**	`research`	—	—
3	**The ACB-4000 personality** — SCSI target + `md10`/`reference` shim profiles	`firmware`	1, 2	#179
4	**Hard-disk volume blob format** — sparse encoding, fill byte, geometry	`tools/docs`	1	#180
5	**MOUNT.COM first-step menu** — personality-aware (floppy / Drive C / hard disk), plus mailbox `KIND=2` and the drive-mapping table	`software/docs`	3, 4	#181
6	**Web-UI functional spec** — the personality registry, image library, status	`docs`	—	#182
7	**CoPower-88 coexistence** — BIOS install order and drive-letter policy (#26, #28)	`software`	3	#183

9.5 What closes #8

- ☒ The command sets of both products documented — the ACB-4000's from its OEM manual (§4); the ACT's **recorded as unobtainable** (§3), which #8 allows.
- ☒ A written recommendation — the ACB-4000 (§2).
- ☒ The ACT's port claim re-derived from its own BIOS and written into `docs/occ1-harddisk-analysis.md` (§3).

10. Open questions

1. **Which copy of the tables a boot uses.** The DPB's values *are* in the archive — static data in `HARD.COM`, the type bytes and descriptors at `0xDD5E`/`0xDD6E` and the DPB at `0xDDB6` (§7.3, `docs/md10-system.md` §9) — and so is the system that boots (`docs/md10-system.md` §1). What is not chased is **who copies the static tables into the running system**: the drive carries its own copy of the geometry and type bytes at LBA 4 (`docs/md10-system.md` §3), the BIOS image carries a fallback, and the two disagree in size (`docs/md10-system.md` §9.1). A `peek` at `0xDD5E`/`0xDD6E` on the running system settles it.
2. **The MD-10 host-adapter bit map** — which observed status bit is `REQ`, which is `BSY`, which is phase (§6). Bounded, because the *protocol* is known; the four bits the driver polls are now read out in `docs/acb4000-host-adapter.md` §1 (`01h`/`02h`/`20h`/`40h`).
3. **"36 user areas"** — a driver extension, or advertising copy?
4. **Is the MD-10's drive-letter assignment reconfigurable?** It decides whether we can leave the floppies at `A:`/`B:` (§8).
5. **Was the MD-10 host adapter a Z80-socket device or a port card?** §5's portability argument assumes the former; no photo of it has surfaced.
6. **The ACT board** — for the record only; the decision does not wait on it (§3).

Sources

- `research/harddisk/` — the ACT software (`OCC1_HARDDISK.IMD`, `extracted/`), disassembled for §3.
- `research/adaptec/` — the MD-10 software (three disk images), disassembled for §6; and the archived ACB-4000 primary sources under `research/adaptec/acb4000/` (§9.1).
- `research/technical-articles/TPC-Issue05-1983-pp25-mega-byte-md10-hard-disk-ad.pdf` (OCR at `…ocr.txt`) — the MD-10 advertisement: sizes, drive letters, user areas, memory overhead (§6, §7).
- `docs/md10-system.md` §9 — the DPB and the drive tables read out of `HARD.COM`, the one-DPB consequence, and the system that boots (§7.3).
- `docs/occ1-harddisk-analysis.md` — the product analysis this decides on top of.
- `docs/o1-memory-io-map.md` §3 — the `2900`/`2A00`/`2C00` device map used in §3.
- `docs/drive-c-protocol.md` §8.3–§8.5, `docs/drive-c-design.md` §4.2 — the CP/M ceiling, the drive-letter precedent and the blob precedent (§7, §8).
- `docs/copower88-protocol.md` — the CoPower-88 host protocol this document's §8 re-orders.
- `docs/virtual-peripherals-block-diagram.md` §5 — the personality table this re-orders.

The RT-60A real-time clock

Status: Partially solved. The register map, the memory ABI and the sentinel rules are settled and are what `firmware/rt60a-rtc/` implements (#7, #18); the RT-60A's own IEEE-488 wake-up handshake is *retired* as a known unknown rather than pending, to be reopened only if a unit or a good `CLOCK.COM` dump turns up.

The RT-60A is a **battery-backed real-time clock that plugs into the Osborne 1's IEEE-488 port** and passes that port through, and the one fact that decides everything below is that **it does not put the time anywhere in memory — `CLOCK.COM` does.** So the thing to build is a *publisher*, not a chip, and the deliverable is a **contract** (§"The contract", #7): a pointer word at **`0x0040`** holding the address of a block whose fields run from `−15` to `+19` (flags, time and date formats, binary `DAY`…`SECONDS`, `MEMBUFF`, and two ASCII strings); a cadence of one `MEMBUFF` decrement per 60 Hz tick on the interrupt vector at **`0xEFF8`** with the block refilled about once a second; and three reader states distinguished by the pointer (`0xFF`/`0xBB` = no clock software) and by `MEMBUFF` (stuck at 1 = stale, not absent). The same publisher also keeps the machine's own cells `EF66–EF69` in step, which is the whole of the "third shape" question (§"The third shape", #81).

Three things are **not** established, and none of them blocks the ABI: the exact IEEE-488 wake-up handshake, which only `CLOCK.COM` speaks; the MSM5832 register semantics, taken from the datasheet rather than from a running chip; and anything that would need the vendor's disk, because the TD0 on bitsavers is a damaged dump (27 bad sectors, extraction failed).

Source: `research/rtc/RT-60A_Manual.pdf` (JG Communications, Tucson AZ) — this doc reconstructs the interface from the manual, on the user direction "virtually rebuild from descriptions".

Hardware

- Plugs into the Osborne 1 **IEEE-488 port** (card-edge), battery backed (DL2420/DL2430/CR2320 lithium, ~1 yr), runs standalone.
- Based on **OKI Semiconductor MSM5832 / MSM58321** CMOS RTCs (BCD time/date chips, 4-bit bus, address-selected registers).
- **Pass-through**: an output connector + extra circuitry keeps the IEEE-488 / Centronics / general-purpose port usable. The clock's outputs are normally **high-impedance**; a "special low voltage CMOS circuit" recognizes a **unique handshake sequence** from the software to activate clock outputs for reading. The handshake is designed not to disturb IEEE/Centronics devices.
- Prerequisite: the port must be in **source-handshake mode** (Centronics output mode). If not found (or an interrupt hits mid-handshake), the read is deferred to the next 1/60 s interrupt.
- Crystal-controlled with a trimcap for speed adjustment.
- Clock reset only by deliberately shorting two vertical pins on the board (prompted by CLKSET).
- Bulletin #2 documents a **Drive C modification** — relevant since Drive C also uses the IEEE port (6821 PIA bit-bang); protocol clash management matters for our interposer coexistence too.

Host software architecture (CLOCK.COM)

- Installed by changing the boot disk to call **CLOCK.COM instead of AUTOST.COM** on both cold and warm boot; OS relocated **1 KB lower** to make room for resident clock code above the BIOS.

- Hooks the **interrupt vector word at 0xEFF8** (O1 60 Hz interrupt; CTC-driven). Each tick decrements MEMBUFF from 60; at 1 it reads the clock (~once/second), translates time to binary + ASCII, stores in registers, chains to the original handler.
- **Phantom Display**: on-screen clock updated every 60th via Z80 block-transfer screen sensing (costs ~15% CPU).
- **At Trap**: patches the BIOS **LIST output jump**; intercepts `@T` / `@D` in printer output and expands to time/date (char after @T/@D must be space or CR).
- File time/date stamping patches installed at boot (BDOS-level).
- Warm-boot reload needed for stamping patches; 1.4x ROM can bypass (bulletin #7).

Software-visible register map (offset from pointer at 0x0040)

Pointer word at **0x0040** points to DAY; add offsets (hex offsets shown relative to the pointer):

Offset	Name	Contents
−15 (FFF1 rel)	FLAGS	bit0: phantom on; bits3,4: at-trap use; bit5: at-trap on; bit7: software off
−14 (FFF2)	TIMEFLAG	bit0: 0=12h/1=24h; bit1: 1=AM/PM off; bit2: 1=seconds suppressed
−13 (FFF3)	DATE (7 B)	YYMMDD, DOW order; bit2 of 10s-of-days = leap year. **Only always-current date rep**
−6 (FFFA)	TIME (6 B)	raw BCD unpacked 12-hour, HHMMSS; bit2 of first byte: 1=PM. Raw from clock chip
0 (0000)	DAY	binary day (SD: FF/BB = invalid)
1	MONTH	binary
2	YEAR	binary (0x53 = 1983)
3	HOURS	binary, 24 h, 0 = midnight
4	MINUTES	binary
5	SECONDS	binary
6	MEMBUFF	60⟶1 decrementer; sticks at 1 when reads failing (port not in output mode etc.)
7	TIMEASCII	NUL-terminated ASCII time string (chosen format)
19 (0013)	DATEASCII	NUL-terminated date string

The contract — what an RT-60A-compatible time service must publish (#7)

This is the *specification*, promoted out of the analysis above. It is what `firmware/rt60a-rtc/` (the resident publisher, #18) and the interposer (#12) must agree on. The rule that makes it implementable is the one sentence that decided ##7: **the RT-60A does not put the time anywhere in memory — `CLOCK.COM` does.** The hardware is read-only; the ABI is software. So we build the *publisher*, not the *chip*.

The pointer. A word at **`0x0040`** holds the address of the block. It is the only fixed address; everything else is an offset from the pointer.

The block (offsets from the pointer, negative offsets below the pointer):

Off	Len	Field	Encoding
−15	1	FLAGS	bit0 phantom display; bit3/4 At-Trap use; bit5 At-Trap on; bit7 software off
−14	1	TIMEFLAG	bit0 0=12 h / 1=24 h; bit1 1=AM/PM off; bit2 1=seconds suppressed
−13	7	DATE	YYMMDD+DOW, BCD; bit2 of tens-of-days = leap year. **The only always-current date**
−6	6	TIME	raw BCD HHMMSS, 12-hour; bit2 of the first byte = PM. Straight from the chip
0	1	DAY	binary
1	1	MONTH	binary
2	1	YEAR	binary (0x53 = 1983)
3	1	HOURS	binary, 24 h, 0 = midnight
4	1	MINUTES	binary
5	1	SECONDS	binary
6	1	MEMBUFF	60⟶1 decrementer; sticks at 1 when reads fail
7	12	TIMEASCII	NUL-terminated ASCII time string, chosen format
19	12	DATEASCII	NUL-terminated ASCII date string

The cadence. CLOCK.COM hooks the interrupt vector **word at 0xEFF8** and decrements MEMBUFF once per 60 Hz tick; at 1 it reads the clock (~1/second), translates to binary + ASCII and refills the block, then chains to the handler it replaced. 0xEFF8 is the stock ROM's keyboard-interrupt

vector: `INTBL = 0xEFF0` and the ROM sets `INTBL+(4*2)` (`rom144.asm` line 235) — so the vector slot is `INTBL` entry 4, and a replacement publisher must save and chain, not simply overwrite.

Sentinel and invalidation rules (the "no clock" vs "clock stopped" question)
These are the rules a reader must be able to rely on — #7 asked for them explicitly, and they are what distinguishes the three states:

State	How a reader tells	What a reader must do
No clock software installed	pointer word at `0x0040` is `0xFF` or `0xBB`	treat time as unavailable; do not POKE `0x0040` yourself
Clock software installed, reads failing	pointer is valid but `MEMBUFF` has stuck at 1 for more than ~2 ticks	time is *stale* — show it, do not trust it for stamping
Clock running	valid pointer and `MEMBUFF` decrementing	`DAY…SECONDS` and `DATE/TIME` are current

`MEMBUFF` is therefore not just a countdown: **it is the health signal.** A publisher that cannot read the clock must leave `MEMBUFF` at 1 (the documented behaviour) rather than let it run free, so that a reader can distinguish "no clock" from "clock stopped". `DATE` at −13 is the one field the manual calls always-current even when the clock read is deferred, so it is the field to trust when `MEMBUFF` is stuck.

What this buys. Any software written against the documented RT-60A ABI — `CLOCK.COM`'s consumers, the `@T`/`@D` At Trap, the BDOS file-stamp patches — works unchanged against our publisher. What we deliberately do **not** reproduce is the undocumented IEEE-488 wake-up handshake, which buys nothing because the only thing that speaks it is `CLOCK.COM` itself.

Conditions that block clock reads (from HELP)

1. Disabled interrupts; 2. peripherals off/off-line; 3. port set as input for extended periods; 4. DAV/ATN held low; 5. printer status lines wired to NRFD (IEEE pin 13) — Centronics cables should have **no connections on pins 13 and 21**; 6. long periods not in IEEE source-handshake mode / long data transfers.

WordStar WSMODS patch (DD systems)
UCNSTA JMP → 0DD06H, INCON JMP → 0DD09H, third JMP → 0DD0CH (SD: 0E106H/0E109H/0E10CH; original DD values 0E506H/0E509H/0E50CH).

Prior art: a *software* date facility, before the RT-60A (#18, #28)
Before any clock hardware, the machine's users kept the date in **page zero**. "**Dating your computer — a system's date facility**" by **Walter B. Novinger**, *The Portable Companion* #7 (June 1983, PDF pp. 50–51 = printed 48–49; extract `research/technical-articles/TPC-Issue07-1983-pp50-51-dating-your-computer-date-in-memory.pdf`), describes a three-part scheme:

- **`DATE.BAS`** prompts `MMDDYY` at boot, validates month/day/year, and pokes the date into **`0050H–005BH`** — a **12-byte block the *Osborne 1 User's Guide* "Memory Allocation" chart (p. R-243) marks "not used"**. It is written twice over so a reader can take either form: **binary** (`POKE 80,D`, `POKE 81,M`, `POKE 91,Y`) and an **ASCII month name** at `82H–8AH`. `DATE.BAS` then `NEW`s itself, leaving BASIC pristine.
- **`GETDATE`** is a BASIC subroutine that reads the block back (`PEEK(80)` … `PEEK(91)`) — no machine code, no `CALL`/`USR`.
- **`AUTOST.COM`** is modified (per the User's Guide ch. 8 listing) so boot runs `MBASIC DATE`: `LXI B,10` → `LXI B,13` (command longer than 8 characters, so `length + 2`) and the command string becomes `'MBASIC DATE',0`.

The buffer survives a **warm start (RESET) and diskette swaps** but **not power-off** — the same "no hardware, so it dies with the power" limitation the RT-60A exists to fix. That is why it is the right prior art for #18: it is what a user did *before* buying a clock, and it fixes the **page-zero ABI** that any emulated date must not disturb. `0050H–005BH` is also worth remembering as a **free 12-byte scratch block** for the interposer (#12) — see `docs/o1-memory-io-map.md` §2.

The third shape: the O1′s *own* clock, and what OZROM 1E does with it (#81)

##7′s stated worry was "read the OZROM manual before we fix our own ABI, so we do not invent a third shape". Reading it — and the OCC source — turned up something better than a warning: **the Osborne 1 already has a native software clock, and OZROM did not invent an ABI either; it adopted the O1′s own reserved cells.** There is no third *invention*; there is a third *convention*, and it predates both third-party products.

The machine already ticks a clock — `UPTIM`

The stock ROM's 60 Hz keyboard interrupt handler calls `UPTIM`, which keeps the time as BCD and handles the bell and disk-active countdown:

```
GKEY:                       ; rom 1.3 source, BMKEY.ASM
    CALL UPTIM              ;UPDATE SYSTEM CLOCK
UPTIM:
    ;Update time via 60hz interrupt
    ;  Time is kept as hhmmss (3 locations)
    ...
    DEC [hl]                ; HL => SEC6  (sixtieths)
    RNZ                     ; if not 1 sec transition
    STO 59,[hl]             ; 60th tick is NOW
    XOR A                   ; A:=0, CY:=0 for DAA
    DEC HL                  ; HL => SECS
    LD A,[hl] ! INC A ! DAA ! STO A,[hl]
    CMP 60h
    RC                      ; if not 60 secs
    STO 0,[hl]
    DEC HL                  ; HL => MINS
    ...
    CMP 25h                 ; 24-hour wrap
```

Measured two ways, and they agree:

- the increment code walks **SEC6 → SECS → MINS → HOURS**, so the cells run downward in address from SEC6; and
- the CBIOS RAM map (occram25.asm) places them with HOURS: DS 6 / MINS=HOURS+1 / SECS=HOURS+2 / SEC6=HOURS+3, and its DACTVE = HOURS+4, BELCNT = HOURS+5 match the ROM's own absolutes DACTVE = 0EF6AH, BELCNT = 0EF6BH (rom144.asm lines 128–129).

So the addresses are fixed:

Cell	Address	Meaning
IDAY / IMONTH / IYR	EF63 / EF64 / EF65	calendar date, binary
HOURS	**EF66**	hours, 2-digit BCD
MINS	**EF67**	minutes, 2-digit BCD
SECS	**EF68**	seconds, 2-digit BCD
SEC6	**EF69**	sixtieths, 2-digit BCD
DACTVE	EF6A	disk-active countdown (ticked by the same routine)
BELCNT	EF6B	bell countdown

The clock **starts at midnight on every boot** (the ROM clears EF00–EFEF), nothing in the stock machine *sets* it, and nothing displays it. It is a running counter with no reader and no setter — which is precisely why both third parties built on top of it rather than beside it.

The ROM 1.3 source's UPTIM comment says the time is "kept in video memory", which is inconsistent with the address map (video is F000+, the clock is EF66+); the addresses are confirmed by two independent files, so the comment is stale, not the map. **ROM 1.44's source does not tick the clock**: its GKEY was rewritten ("fixed bell bug", rom144.asm line 16) and now handles the bell and DACTVE but no longer decrements SEC6, and the file contains no SEC6/UPTIM reference at all. So under a 1.44 machine the EF66–EF69 cells are *reserved but not ticked.*

OZROM 1E adopts those cells and adds the setter, the display and the hook

OZROM's documented memory map (manual Appendix C-5) is the same cells, named:

OZROM's name	Address	**the O1 cell**
Start of system clock — hours, 2-digit BCD	EF66	HOURS
Minutes, 2-digit BCD	EF67	MINS
Seconds, 2-digit BCD	EF68	SECS
Sixtieths of a second, 2-digit BCD	EF69	SEC6

and it adds EFDF (time-display toggle), EFE8 (clock position on the first two display lines), and its own interrupt hook at **EFF2–EFF4** (preset to RET, 0C9h) with 192 bytes of interrupt-routine RAM at FCC0–FD7F. OZROM's TIME sets the clock, ctrl-shift-T shows it, and ALARM/TIMER/QUIKTIME ride the hook. **So OZROM's "time-of-day clock" is the machine's own EF66–EF69, finally given a setter and a reader.** Its hook is *not* the RT-60A's 0xEFF8: it is a fixed call OZROM's own SYSINT makes with registers already saved, at EFF2, which is a different INTBL slot and a different calling convention. Full map: docs/ozrom-1e.md.

What this means for our ABI (the #7 decision)
Three conventions now, and they do not conflict:

Convention	Cells	Who reads it
RT-60A / CLOCK.COM	pointer at 0x0040 → binary DAY…SECONDS + ASCII + MEMBUFF	software written to the documented RT-60A ABI
O1 native / OZROM	fixed BCD EF66–EF69 (+ EF63–EF65 date)	OZROM's display/ALARM/TIMER, and anything that reads the machine's clock cells
(ours)	**both**, published together	whichever the software expects

The cheap, correct move is that our resident publisher **maintains the native EF66–EF69 cells as well as the 0x0040 block**. They are four bytes apart in effort and they make our clock visible to OZROM-style readers *and* to RT-60A software, without inventing anything. That is the whole of the "third shape" question: not a new ABI, but two existing ones kept in step.

- We can emulate the RT-60A faithfully WITHOUT its exact hardware handshake: implement the MSM5832 register semantics and the 0x40 pointer ABI, plus the IEEE-port handshake activation observed at the PIO lines. For software compatibility, matching the **memory ABI + interrupt cadence + At Trap/BIOS hooks** matters more than the internal CMOS circuitry.
- MSM5832 register set (from datasheet, 4-bit regs): sec 1/10, min 1/10, hr 1/10 (with AM/PM + 12/24), day-of-week, day 1/10, month 1/10, year 1/10 — matches the BCD layouts above.
- Drive C coexistence: both devices live on the same IEEE port; the interposer can present BOTH virtual devices with proper arbitration (Drive C's 6821 bit-bang vs clock handshake) — worth an interference test in #29.

CoPower-88: the host protocol

Status: Partially solved. The port pair, the handshake and the block transfer are read out of the two RAMDISK.COM builds and corroborated by the board's own schematic; what is left is the 8088-side command set — the monitor ROM is in the repo (research/roms/swp-p88.rom, #70) but has not been disassembled for it — and the meaning of every status bit other than the two the drivers test.

The CoPower-88 is an **8088 co-processor board sitting on the host Z80′s bus**, and it presents the host with a **two-port mailbox**. The **even** address of the pair is the data/command port; the **odd** address is status. Every transfer is a poll of the status port followed by an `OUTI`/`INI` block. The pair is configurable twice over, and both have to agree: the board takes it from an 8-position switch (`docs/copower88-schematics.md`, §"Port decode"), and the driver is told the same pair at install time, defaulting to the two bytes in the `.COM` header.

Serves #3 (the Z80 socket and pinout this board sits in), #4 (the CoPower-88′s own ROM, which is in the archive), #5 (the port decode) and #23 (the virtual CoPower-88 personality this contract is written for).

Port	Address	Direction	What it is
Data / command	even — `D` in the driver (Kaypro `7Eh`, Zorba `FEh`)	both	`OUT` latches host⟶8088; `IN` reads the 8088⟶host latch
Status	odd — `E` in the driver (`7Fh`, `FFh`)	read	bit 0 **set** = the host must wait before a write; bit 7 **set** = a byte is waiting for the host
Doorbell	the same odd address	write	sets `INTR` on the 8088 (sheet 4). **Neither RAMDISK driver uses it**: every write they make goes to the even port, and that write is itself the doorbell

The pair is stored as one word at `0107h` — low byte (`E`) = status, high byte (`D`) = data — so a port override has to keep that order. `RAMDISK.COM` asks for it directly: `CoPower_88 port address or <CR> to use default`.

Sources: `research/copower88/kaypro/extracted_cpm/` (copwrcpm.td0) and `research/copower88/zorba/extracted/` (swpdos.td0). Disassemblies in `*/disasm/` (z80dasm, org 0x100). Schematics: `research/copower88/zorba/611-0003-{1..4}.pdf` — the port decode is settled in `docs/copower88-schematics.md` (#5).

The transfer protocol, instruction by instruction

Both drivers carry the same subroutine at `0B82h` (host ⟶ 8088) and `0BE9h` (8088 ⟶ host) — the machine-specific parts of `RAMDISK.COM` are the port defaults and the install-time patching, not the transfer:

Step	The code	What it means
1	`LD DE,(0DB6h)` / `LD C,E`	load the port pair; `C` := the **status** (odd) port
2	`IN A,(C)` / `RRA` / `JP C`	**wait while status bit 0 is set**
3	`LD C,D` / `OUT (C),A`	write the command byte to the **data** port
4	`LD A,005h` … `LD A,001h`	the two-byte preamble `05h`, `01h`, each behind a fresh poll
5	`OUTI`, with `LD B,080h`	the block, **128 bytes at a time** — one poll per byte
6	`IN A,(C)` / `JP P` / `LD C,D` / `IN A,(C)` / `CP 005h`	on a read, **wait while status bit 7 is clear**, then expect `05h`
7	`CP 001h`, then `INI`	the same `05h`, `01h` preamble, then the block read

Three consequences worth stating plainly:

- **`05h`/`01h` is a preamble, not a direction code.** Both directions open with it, so it frames the transfer rather than selecting it; what distinguishes a read from a write is which subroutine the driver called. The length lives in the driver's own work cell (`0CBDh`), and the `080h` in the block loop is CP/M's 128-byte record.
- **The status port is read for two different bits at two different polarities** — bit 0 *set* means wait before a write, bit 7 *clear* means wait before a read. The schematic accounts for one host-visible flag, bit 0, toggled by the 8088′s write (`docs/copower88-schematics.md` §"Mailbox registers"); what drives bit 7 is not shown on the sheet.
- **The header bytes are the configuration** (identical in both images): `0103h` = the assigned drive number, zero-based — `SUB 41h` at `037Bh`, `ADD A,41h` at `0753h` — default `0Ch` = drive M; `0104h` = the erase-directory answer (`59h` = `Y`); `0107h`/`0108h` = the port pair above. `RAMDISK` asks for all three at install time: drive letter (A–P), erase-directory Y/N, load address (page-aligned, `??00`) and the port pair, then installs a BIOS wedge — *"not standard CP/M system, cannot install"* if the BIOS jump table is not where it expects it.

Machine glue differences (Kaypro MSDOS.COM vs Zorba PCDOS.COM)

- Kaypro MSDOS.COM touches Kaypro motherboard system ports (0x10-0x14 SIO/CTC/system, 0x1C/0x1D/0x1F heavily) IN ADDITION to the CoPower ports — screen/keyboard/clock glue for the "BIOS calls only" PC emulation environment.
- Zorba PCDOS.COM error strings show the watchdog protocol: "INVALID 8088 REQUEST #XX AT XXXX", "8088 DOES NOT RESPOND", "8088 SYNC ERROR, PRESS R TO RETRY" — the Z80 host polls the 8088′s request queue; the 8088 ROM monitor services requests asynchronously.

- Both drivers are the same vintage SWP codebase (identical io subroutine offsets) — the machine-specific parts are: port defaults, console/screen BIOS glue, and install-time BIOS vector patching.

Implications for the Osborne 1 port (#26)

1. **The Osborne board's pair is not a free choice, and it is not the Kaypro default.** The CoPower-88 decodes the pair on its own board — it snoops only `IORQ`, one address bit and the data bus (`docs/copower88-schematics.md`) — but the host's decode sees the same write cycles, and the O1 decodes **only `A1:A0`** of an `OUT` (`docs/o1-memory-io-map.md` §4, measured), so every port aliases one of `00h–03h`. The member that matters is the **even** one, because that is the one written on every byte: `7Eh`/`FEh` alias `02h` = *set `BIT 9`*, which maps the dim-bit bank over video. A port override therefore has to land on a pair whose written address is harmless under `A1:A0` — `00` (bank-2 select, which the machine re-derives) or `11` (clear `BIT 9`, idempotent) — never `10`. `docs/mailbox-protocol.md` §2.1 dodges the same hazard by the same rule, and `docs/o1-memory-io-map.md` §7′s "avoid `0x00–0x03` entirely" is the general form of it. **What the pair is set to on an Osborne board is a measurement, not a guess** — see "Open questions".
2. What *is* settled is that a port override needs no change to the driver's code: the pair is one word at `0107h`, so once the board's setting is known the existing driver talks to it with a different constant.
3. MSDOS.COM's Kaypro system-port access must be mapped to Osborne equivalents (bank-2 memory-mapped ACIA/PIA) — the BIOS-call translation layer is where the porting effort lives.
4. The 8088-side firmware is on the board's ROM (not on these disks); the DOS disks (copwrdos.td0, dos211-*.td0) are PC-side payloads. That ROM is in the repo as `research/roms/swp-p88.rom` (#70) — 8088 code, dumped from an SWP board.

Open questions

- **The 8088-side command set** beyond the `05h`/`01h` preamble — sector read/write, identify, sync. `research/roms/swp-p88.rom` (#70) is that monitor, in 8088 code and already in the repo, so the answer is a disassembly away.
- **What drives status bit 7**, the read-side "data available" flag. Sheet 4 shows one host-visible flag, on bit 0; the read path waits for bit 7.
- **The Osborne variant's port pair** — and whether it could keep the Kaypro default at all, given the `A1:A0` alias above. The board photographed is a KAYPRO-88 (`docs/photo-survey.md` §"CoPower-88 8088 board" — the photo drop for **#9**), and that document reads the base as a comparator with a hardwired reference rather than the 8-position switch the drawing shows; the drawing is the primary source here, and no Osborne board's setting is recorded either way.
- **The other six status bits** — the drivers never test them, and the schematic drives only one.

CoPower-88: the schematics

Status: Solved. The four sheets settle what the board is and how the host talks to it: an **8088 in minimum mode** with its own clock, RAM and refresh, and a **switch-set port decode into a two-port mailbox** with a doorbell. What the sheets do not settle is the second status bit the

drivers poll before a read (bit 7, §"Protocol synthesis") and the base the *Osborne* variant's decode is set to.

The board is the closest thing in the archive to this project's own interposer: it sits on the host's `IORQ` space with a **minimal signal set** — `BIORQ`, `RDB`, `WRB`, one address bit and eight data lines — and it seizes the host bus with `HOLD` when it needs to. Its host interface is a two-port mailbox whose **base is set on the board, in units of 2**; the protocol that mailbox speaks is read out of the drivers in `docs/copower88-protocol.md` (#5), and the two agree. That matters here twice: as prior art for our own interposer (#16/#19), and as the board a CoPower-88 personality would have to emulate (#26).

Sources: `research/copower88/zorba/611-0003-{1..4}.pdf` ("SCHEMATIC 8088 BOARD", Software Publishers Inc., Arlington TX).

Sheet 1 — 8088 CPU
8088 (U11) in minimum mode (MN/MX hi), 8284 clock generator (U5) with X'TAL1 + reset conditioning, address demux via LS373 (U19, A0-A7) and LS375 (U12, A16-A19), data bus transceiver LS645 (U20, DIR/EN), control buffering via LS244 for RD/WR/IO-M/DEN. HOLD/HLDA + INTA/INTR brought out. So: standalone 8088 system, 20-bit address space.

Sheet 2 — RAM and ROM array
16 x 4164 = 128 KB DRAM on-board ("M1 THRU M16"), header "TO 128K RAM EXPANSION BOARD" (256 KB total option). 2732A 4 KB boot/monitor ROM (U28) with ROMCS + RD/OE gating. Address muxing via LS257 (U26/U27), refresh address counter LS244 (U18) driven by REF1-7/REF-ENA. ROM data buffered by LS244 (U23).

Sheet 3 — Memory controller
Refresh timing: LS393 (U4) + LS161 x2 (U9/U10) generate REF1-REF7 from CLK under REF-RST. Refresh arbitration state machine (LS175s U2 + LS153 U3) arbitrates CPU vs refresh, generating CPU-ENQ, REF-ENQ, RFSH and — critically — **HOLD**: the board asserts HOLD to seize the HOST Z80′s bus during refresh/arbitration windows. RAS decode: LS138 (U31) CAS0-CAS3, LS138 (U32) + LS08 (U33) RAS0-RAS3 + ROMCS, qualified by IO/M, A16, A17, A19, HLDA. MUXC + LS74s (U30) for address mux phase.

Sheet 4 — Z80/8088 PORTS (the host interface)
Host connector P1 carries: BIORQ, RDB, WRB, BA0 (Z80 A0), BD0-BD7 — i.e. the board snoops ONLY IORQ cycles, one address bit, and data.

Port decode
- 4077 XNOR comparators (U3) + 4021 shift register compare A1-A7 against an 8-position **DIP switch** (100K networks) ⟶ DECODE. **The I/O base address is switch-selectable in units of 2**, and it is set on the board rather than by software. This matches RAMDISK.COM's "CoPower_88 port address or " prompt: Kaypro factory default 0x7E/0x7F, Zorba 0xFE/0xFF. (The one board photographed reads that package as a comparator with a hardwired reference

instead — `docs/photo-survey.md` §"CoPower-88 8088 board". The drawing is the primary source; the Osborne variant's setting is unrecorded either way.)
- LS139 (U6): BIORQ + DECODE + BA0 → PORT0 (A0=0) / PORT1 (A0=1).

Mailbox registers
- U24 + U25 (LS373/LS373): two 8-bit latches forming a bidirectional data mailbox between BD0-7 (host) and D0-D7 (8088). PORT0 WRB latches host→8088 data (also clocking D0 flag LS74 U7 → STATIN visible to 8088); RDB+PORT0 reads the 8088→host latch.
- **PORT1 write sets INTR (LS74 U14) → interrupts the 8088**; cleared by 8088-side ack / RESET. This is the "command doorbell": host writes status/command port to kick the 8088 monitor.
- IRQOUT (WR + LS32 U15): 8088-side write toggles the host-visible status flag; host polls STATIN (RD + LS32) — **bit 0 of the status port**, exactly the `rra / jp c` poll loop in the drivers.
- RDX (from RD8088 via 40HC32 one-shot-ish network) heads to "PIN 21 OF BERG PINS" — read-strobe toward the RAM expansion header.
- HC245 (U1): 8088-side data buffer BD0-7 ↔ D0-7 (daughter board).

Protocol synthesis (hardware + drivers agree)
1. Host polls PORT1 (the odd port) and writes command byte(s) to PORT0 (the even one). The drivers test **bit 0** before a write — `IN A,(C)` / `RRA` / `JP C`, waiting while it is *set* — and **bit 7** before a read, waiting while it is *clear*. Bit 0 is the flag this sheet accounts for; what drives bit 7 is not shown here (read out of the drivers, `docs/copower88-protocol.md`).
2. 8088 monitor (2732A ROM) services the request, fills the 8088→host latch, toggles IRQOUT → status bit.
3. Host sees the flag change, streams data via OUTI/INI on PORT0. The two bytes RAMDISK.COM sends — 0x05 then 0x01 — are a **preamble** that opens the transfer in both directions, not a direction code.
4. **Two ways to ring the doorbell, and the drivers use the quieter one.** A write to PORT1 sets INTR (LS74 U14) outright; a write to PORT0 also clocks the D0 flag (LS74 U7) that the 8088 can see. Every write in both RAMDISK.COM builds goes to PORT0, so the 8088 is kicked without INTR being raised at all.

Implications for the interposer (#16/#19)
- The CoPower-88 is itself a **bus interposer precedent**: it sits on the host IORQ space with a minimal signal set (IORQ, RD, WR, A0-A7, D0-D7, RESET) and steals bus cycles via HOLD for refresh. Our RP2354B design needs the same signals, plus WAIT for finer-grained interposition (the SWP board couldn't stretch cycles; it used HOLD).
- Emulating a CoPower-88 (#26): trap IORQ at the switch-configured base, implement PORT0/PORT1 with both status bits (bit 0 write-side, bit 7 read-side), and run the 8088 side virtually (or stub the monitor protocol).
- For finding the Osborne CoPower-88's switch setting without a schematic: scan even/odd port pairs for the mailbox signature (write PORT0, read back after 8088 echo; or observe INTR behavior). MSDOS.COM's "8088 DOES NOT RESPOND" path gives a detection oracle — and

any such scan has to respect the host's `A1:A0` alias, because a write to a port `≡ 2 (mod 4)` sets `BIT 9` and maps the dim-bit bank over video (`docs/o1-memory-io-map.md` §4).

80-column boards: SCREEN-PAC to TRA

Status: Partially solved. Four 80-column implementations are identified, and each one's attachment, width selection and detection is read from its own material; what is left is one measurement (§"How software finds the board") and Osmosis's *technical* manual, which the archive does not hold.

Four 80-column upgrades for the Osborne 1 are known, and **no two of them attach, select a width or announce themselves the same way**. That is the fact a video personality (#24) has to be built around, and it is why "80 columns" is not one interface:

Board	What it actually is
OCC SCREEN-PAC (1983)	an add-on board inside the machine, hardware only — the stock BIOS already carries the width routine, so no ROM changes
The Research Associates' TRA (1983)	an external video unit that reaches the monitor through the **IEEE-488 port**, with the printer moved to a parallel port on the TRA
Nuevo (1984)	a **second video system**: a replacement monitor ROM, a CBIOS patched on disk, and its own Z80, CRTC, video RAM and character generator on the board
Osmosis (1983)	a **Z80 interposer board** — the CPU moves onto it and it takes the CPU's socket — with a terminal-style escape-sequence interface and a supplied monitor ROM

Nuevo Electronics and Osmosis each shipped **two** Osborne 1 upgrades — a double-density board and an 80-column video board — and **neither follows the pattern OCC used for the same two functions**. That matters twice over: it is prior art for this project's interposer, and it means a video personality (#24) cannot assume the ScreenPac's interface.

This doc covers the **video** side. The DD side is in `docs/dd-upgrade-formats.md` and `docs/research-gaps.md` gap #11.

"Executive" is a different machine, and treating it as an Osborne 1 is the one mistake that corrupts every conclusion below. In Osborne's world the Osborne **Executive** is the O2-class machine, not a variant of the 1, so upgrades, software and advertisements aimed at it are **not Osborne 1 evidence** — for example the $595 double-sided-drive upgrade advertised in *The Portable Companion* ##14, PDF p. 47, which is an Executive product. The three machines this

project cares about are the Osborne 1, the Osborne 1 with OCC's own upgrades, and clones of them (`docs/research-gaps.md` gap 12).

The four implementations at a glance

	OCC **SCREEN-PAC** (1983)	**TRA** (1983)	**Nuevo** (1984)	**Osmosis** (1983)
What it is	add-on board, hardware only	external video unit with its own parallel printer port	replacement monitor ROM + patched CBIOS + own char-gen	**Z80 interposer board** + supplied monitor ROM
How it attaches	board inside the machine (`2F00040` service manual §6.12)	**IEEE-488 port (P3)** and an external monitor	swap the monitor ROM; patch the CBIOS with DDT	**Z80 moves onto the board, board into the Z80 socket**
ROM change	**none** — MAME models `osborne1sp` as a clone	none mentioned — vendor-supplied driver software	Nuevo **1.51** monitor ROM + BIOS 1.5 objects	supplied ROM marked **"Osborne 1.44 GM"**
Widths	**52 / 80 / 104**	80 and 100, windows into the same 128×32 format	80	80 (and back to 52)
Selected by	**BIOS variable** — poke `E1A5H`, call `E4E5H`	the driver, after a "one-time, automatic disk configuration process"	patched CBIOS entry (`02C9` → `JP 02D4`)	**escape sequences** — `ESC 8` / `ESC 5` / `ESC F`
Detected by software	`2400H` bank 2, **bit 2 LOW**	— (not documented in the ad)	BIOS 1.5 reads a port the O1 does not decode	— (no register; the probe *is* a command)
Horizontal scroll	32-column blocks	soft scrolling	—	—
Character generator	stock late	—	**its own** (adds underline via row 9)	stock 1.44

Sources for each row are quoted below. "—" means the material in the archive does not say; it is not a claim that the feature is absent. The **TRA** column is the exception: that board is not in the archive at all, so its row comes from its advertisement alone (below).

OCC SCREEN-PAC — hardware only, driven by the stock BIOS

The one that needs no ROM: `docs/o1-memory-io-map.md` §2a has the full statement. The parts that make it the odd one out are that the **stock BIOS already contains the width-switching routine** (`E1A5H` / `E4E5H`), that the board announces itself at **`2400H` in bank 2**, and that the SCREEN-PAC's own 2 KiB ROM is *the same image as the stock late character generator* (`docs/o1-rom-variants.md`) — so the capability genuinely is not in a ROM.

The Research Associates (TRA) — 80/100 columns, over the IEEE port

The only one of the four that reaches the screen through a *different connector* — and the only one the archive does not hold. An advertisement in *The Portable Companion* #5 (February/March 1983, PDF p. 98 = printed 84; extract `research/technical-articles/TPC-Issue05-1983-pp98-tra-80-100-column-video-and-winchester-ads.pdf`) announces "**80 and 100 column video**" from **The Research Associates** (Forest Hill, Dunstable, Massachusetts) as "an add-on to the Osborne 1", providing:

- an **80×24 window** and a **100×24 window** "into a 128 character by 32 line video format" — the same 128-wide buffer this project already models (`docs/o1-memory-io-map.md` §2a), so TRA widens the *window* exactly as the others do;
- **soft scrolling**, **reverse video and blinking**, **composite and TTL video monitor outputs**;
- connection "by any Osborne 1 user. **Simply plug into the IEEE port** and into your external monitor";
- "**parallel printer connection is not forfeited**" — the printer moves to a **parallel port on the TRA**;
- "a one-time, automatic disk configuration process", with the **driver software supplied by the vendor**.

Two things make it worth recording. First, **the interface is the IEEE port**, not the video bus — so a video personality (#24) cannot assume an 80-column upgrade lives on the video side of the machine, and the IEEE port becomes contended between a video device, Drive C (#6/#84), the RT-60A (#18) and Trantor storage. Second, its **on-board printer port** is the same "duplicated port" pattern as the Drive C panel (`docs/drive-c-protocol.md`): the add-on *re-presents* an existing connector. Neither its width-switching mechanism (no ROM or BIOS patch is mentioned — a driver plus a disk configuration step) nor its presence detection is documented in the ad, and **unlike SCREEN-PAC and Nuevo it is not in the archive**: the ad is all we hold.

Nuevo — a new monitor ROM, and a CBIOS patched on disk

Nuevo's video upgrade is **software**, and the archive holds the installer. The system disk `research/nuevo/OS1NUEVO.IMD` (Osborne DD: 40 tracks × 5 × 1024 B, 250 kbps MFM) contains 19 files, and two of them are the installers — not SUBMIT scripts despite the extension, but **DDT patch scripts** run through `XSUB`:

```
$ cpmcp -f osborne1 OS1NUEVO.IMD 0:SYS-80C.SUB -
B:XSUB
B:DDT B:MOVCPM.COM      ; patch the CP/M system image in memory
IBIOS15.OBJ             ; insert Nuevo's BIOS 1.5 object
```

```
R1F00                   ; ...at 1F00
IBIOS15BM.OBJ           ; and the second module
R29E0                   ; ...at 29E0
S2C9                    ; set the byte at 02C9
C3                      ;   to C3 (JP)
D4                      ;    D4 02 = JP 02D4  <- the 80-column hook
02
.
FA10,A17,00             ; clear the width/variable block
G0
SAVE 43 MOVE-CPM.COM    ; 43 pages = the new system
MOVE-CPM 59 *           ; write it to the system tracks
```

So the Nuevo mechanism is: **patch `02C9` to jump into new code at `02D4`, carrying a replacement BIOS (1.5) and its variables.** `SYS-DDEN.SUB` is the same script with the DD patches added — four sites set to `64` (100 decimal, the width value used for the wide display elsewhere) — plus an `ERA AUTOST.COM`.

Two more files patch an application rather than the system:

```
$ cat DB23PTCH.SUB
XSUB
DDT B:DBASE.COM
S130 / 01        ; patch dBASE's screen handling
S154 / 50        ; 50 hex = 80 columns
S395A / C9       ; RET
G0
SAVE 70 B:DBASE.COM
```

That is byte-for-byte the same three-byte patch Osmosis's manual gives for dBASE (below). Two independent vendors arrived at the same fix, because what needs patching is dBASE's assumptions, not the board.

Plus the ROM side: Nuevo **1.51** is an OCC ROM carrying Nuevo's copyright and patches (`docs/o1-rom-variants.md`), and the Nuevo char-gen differs from stock — it forces row 9 to `0xFF` when bit 7 of the character code is set, which is the underline attribute made real in the font.

The Nuevo product line, from its own advertising

The Portable Companion #15 (Jan/Feb 1985, PDF p. 39) advertises four upgrades, and the 80-column one is not just a wider window:

> **80 COLUMN UPGRADE** — Increase your screen size to a full 80 columns (& still be able to scroll horizontally if you choose). Excellent resolution, with a **choice of six cursor styles and reverse video capability** for easy viewing. Optional **baud rate generator provides 16 baud rates from 50 to 19.2K baud**. $250.00 (add $30 for the optional baud rate generator)

Three things worth taking from that:

- **"Six cursor styles"** — Nuevo's video board treats the cursor as a *feature*, which is the same circuit the block-cursor modifications (#24) were chasing on the stock machine and on the SCREEN-PAC. A personality that renders only the O1′s underline cursor is rendering the worst of the available options.
- **Horizontal scroll in 80 columns** — the same window-sliding behaviour the SCREEN-PAC gives, here described as a user choice rather than a side effect.
- **A baud generator to 19.2K** — well beyond the 9600 Hogan's modification reaches on the stock board (`docs/o1-memory-io-map.md` §2a), and an *option* rather than a cut track. It is the other end of #21′s range.

The disk products from the same advertisement are in `docs/dd-upgrade-formats.md`.

The Nuevo boards themselves (2024 photo drop)

No Nuevo schematic is in the archive, so both products have only ever been known through their software. Two field photographs change that: they are the **first hardware views of either Nuevo board**. Source: **Mike Loewen**, private Facebook group, 2024-10-14 (post); archived as `research/nuevo/photos/`.

80-column board — silk `NUEVO ELECTRONICS CORPORATION © 1984`, `80 COLUMN UPGRADE REV. D`, **serial 4348**. It is not a video tap; it is a **self-contained video subsystem**:

Device	What it is
NEC D780C-1 (PI9455-138)	a Z80 — the board has its own CPU
Nuevo `SY6545A-1` (8325)	a 6545/6845-class CRTC, so the board makes its own video timing
`SRM2016C25` ×2	2 K × 8 SRAM — about 4 KB of video RAM
socketed **`CHARACTER GENERATOR`** ROM, `88332 SINGAPORE`, dated **6-29-84**	the board's own font, under a Nuevo printed label
`NYMPH 12.288 NMP122`	12.288 MHz oscillator (dot clock)
`74S157` ×3, `74LS194A`, `74LS161B`, `74LS273`, `74LS374B` ×2, `74LS166`, `74LS174`, `74LS86B`, `74LS74A`, `74LS20`, `74LS10`, `74LS02` ×2, `74LS00`, `7406` ×2	glue: multiplexers, counters, shift registers, latches, drivers
`VR1`, `Y2`	video-level trimmer, crystal
`J1`–`J6`, **`J9`–`J12`**	configuration jumper blocks
`P2`, `P3` (10-way edge fingers), `P4`	host/display interface
`BAUD` section, 14-pin `P2111-2`	the **optional** baud-rate generator the advertisement prices at +$30

Two consequences, and both *corroborate* what the software already said:

- **The 80-column mode is a second video system, not a wider window.** Its own CPU, CRTC, video RAM and font are why the advertisement can sell "six cursor styles" and horizontal scroll as *features* — those are properties of a real CRTC. It is also why `docs/o1-rom-variants.md` §"one late image, five names" concludes Nuevo's 80-column design is not the OCC one: the SCREEN-PAC widens the *existing* video path, Nuevo builds another one.
- **The socketed `CHARACTER GENERATOR` chip is dated 6-29-84** — the date in MAME's `character_generator_6-29-84.14`, the Nuevo-era char-gen already tracked in `docs/o1-rom-variants.md`. So the font the board carries is an image the project holds, not an unknown ROM.

Double-density board — silk `NUEVO ELECTRONICS CORPORATION © 1984 MADE IN USA`, `DAL DENS` (dual density), **serial 20011**. This one is an *encoding* board, not a controller:

Device	What it is
`STD DRIVE` silk; `J1`/`J2`; 34-pin IDC header (pins 1–34)	the drive-cable side of the interface (the O1′s own 34-pin floppy header is `P8`)
`WRITE PRE COMP` silk; `74LS195A` (U4), `SN74LS14N` ×2 (U3, U5), `SN74LS00N` ×2 (U2, U6), `SN74LS02N` (U7)	a discrete **write-precompensation delay line** (shift register + Schmitt inverters), and the **data separator**
U1, 16-pin DIP marked `VD9216 / CA00 / 8402`	unresolved. `9216` is the number Standard Microsystems uses for its **FDC9216 floppy data separator**, which would make U1 the separator and put the precompensation in the LS chain — but the marking needs a higher-magnification re-read before either reading is asserted
`RN1`, `RN2`; `P4`	resistor networks; 4-pin power

The likely division of labour — that the DD upgrade supplies the **MFM data separator and write precompensation** the stock machine lacks, and leaves the controller alone — fits the rest of the archive: the controller is `MB8877` ≡ `WD1793` on the mainboard, and the separator is a *separate board on the drive side* rather than part of it (`docs/o1-mainboard-schematic.md` §2, sheets 9/9 and 2/3, U8 and U12–U22). **That is an inference, not a finding** — Nuevo's own installation instructions are the missing piece.

Does any of this bear on Drive C? No. Both boards are host-side — video, and floppy encoding — while Drive C is an IEEE-488 peripheral that draws power from the EXT. VIDEO connector and speaks over the GPIB port (`docs/drive-c-protocol.md`). The photo drop is recorded here because it is the first hardware view of either Nuevo product, not because it changes the Drive C contract.

The 80-column board's own feature list, from the shipped software

`OS1NUEVO.IMD` also carries **`AUTOST.COM`** — a two-instruction CP/M program (`LD C,9` / `CALL 5`, print a string, exit) whose string is the product's own feature list. It is the **autostart banner**: the

80-column installer keeps it (`SYS-80C.SUB`) and the DD-only installer *erases* it (`SYS-DDEN.SUB`), so it ships only with the video product. The same file is on the disk twice, as `AUTOST.COM` and `AUTOST.BAK`, byte-identical.

> **THIS IS AN EIGHTY (80) COLUMN DISPLAY FOR THE OSBORNE ONE.**
>
> FEATURES: (1) EIGHTY COLUMN STANDARD DISPLAY. (2) PROGRAMMABLE BAUD RATE GENERATOR (16 RATES, 50 TO 19.2K BAUD). (3) FULL SCREEN REVERSE VIDEO CAPABILITY (JUMPER SELECT). (4) CURSOR PROGRAMMABLE AS UNDERLINE OR REVERSE VIDEO BLOCK. (5) CURSOR PROGRAMMABLE FOR FAST, SLOW, OR NO BLINK. (6) REVERSE VIDEO BLOCK CAPABILITY (INSTEAD OF UNDERLINE; JUMPER SELECT). (7) COMPOSITE VIDEO OUTPUT FOR EXTERNAL MONITOR (NO MORE ADAPTERS). (8) SIMPLE USER INSTALLATION. — `AUTOST.COM`, `research/nuevo/OS1NUEVO.IMD` (extracted with `cpmcp -T raw -f osborne1`; the manual's BIOS 1.4⟶1.5 list item 3 is the "autostart command space", so the file name is the feature)

This is the product describing itself, and it splits **hardware** from **software** cleanly. Only item **(1)** is the CBIOS patch. Items (2)–(7) are properties of the board's own circuitry: the CRTC gives the programmable cursor (4, 5), the `J1–J6` / `J9–J12` jumper blocks give the reverse-video options (3, 6), the `BAUD` section and 14-pin `P2111-2` header give (2), and the board's own video output gives (7). That is what "a second video system, not a wider window" means in practice — and it is why the advertisement can sell cursor styles and horizontal scroll as *features* while the software delivers exactly one of the eight.

Osmosis — a Z80 interposer with an escape-sequence interface

Osmosis's 80-column board is the one that matters most to this project, because of *how it attaches*. From `research/osmosis/Osmosis_80_Column_Board.pdf` §3:

> Locate the monitor ROM (fig 2.1). This ROM can be carefully removed … Now take the ROM we supplied … Fit the ROM we supplied (marked **Osborne 1.44 GM**) … Locate the Z80 processor … Remove the 40 pin Z80 … **Fit the Z80 into the blue 40 way header on the Osmosis 80 column board** … Fit the Osmosis 80 column board onto the Osborne PCB by inserting the 40 pins … into the socket on the Osborne PCB from which you have just [removed the Z80].

The CPU is moved onto the board and the board takes the CPU's socket. That is a Z80 bus interposer — the same technique this project's interposer uses (#11/#12) — which is why #73 calls Osmosis direct prior art. It is also why the board can be ROM-based: it ships its own monitor ROM.

Its interface is a **terminal's**, not the BIOS's. From §4, the applications guide:

> B3. Press escape (ESC), 8 and return. The screen should now be showing a full 80 columns. … B5. Try pressing escape (ESC), 5 and return to take you back to 52 columns.

and, for software:

> B7. … The board will react to a command such as: `10 PRINT CHR$(27)+"8"`

So Osmosis switches width with **escape sequences** (`ESC 8` = 80 columns, `ESC 5` = 52, `ESC F` = auto-CR/LF) — the shape of a real terminal's control codes, reachable from BASIC, rather than a memory variable. The manual also notes that the board "has been well proven in Europe".

The manual's own dBASE instructions are the same patch Nuevo's disk applies:

> (c) Enter S130 (ret) … Enter 01 (ret) … (e) Enter S154 (ret) … Enter 50 (ret) … (g) Enter S395A (ret) … Enter C9 (ret)

The escape-sequence set (Appendix E)

Appendix E is the *complete* control interface — every code is `ESC` plus one character. It is a display-type table that tesseract renders as noise (the `.ocr.txt` interleaves the columns), so it is **hand-transcribed from the PDF** here. * in the first column is the manual's own mark for "unique to the OS/5 80-column OSBORNE 1":

OS/5 only	ASCII	Hex	Dec	Effect
✳	-	2C	44	sets chars to inverse
✳	,	2D	45	sets chars to normal
	(	28	40	sets full intensity
	)	29	41	sets half intensity
✳	*	2A	42	sets 3/4 intensity
✳	+	2B	43	sets 1/4 intensity
✳	.	2E	46	normal character set
✳	/	2F	47	alternate character set
✳	0	30	48	select default 24-line screen with up/down 48-line scroll
✳	1	31	49	reserved
✳	2	32	50	reserved *(hand-annotated "NO scroll")*
✳	3	33	51	reserved *(hand-annotated "scroll 9 lines only")*
✳	4	34	52	reserved
✳	5	35	53	**display 52 characters**
✳	8	38	56	**display 80 characters**
✳	C	43	67	reserved
✳	[illegible]	[illegible]	[illegible]	[illegible] *the original)*

The source table has two defects. Both are recorded rather than repaired, because they are what the original prints:

- **† the `Dec` column is damaged.** From `R` (52H) to `Z` (5A) the original prints `8-` — the page edge cuts the column — and it prints `183` for `g` where 67H is 103. The values marked † are the correct decimal for the hex shown, not a reading of the original.
- **One row disagrees with itself.** The ASCII column prints `I` (invert screen) and `N` (normal screen); the Hex and Dec columns print `61H/97` and `62H/98`, which are `a` and `b`. Every other row's three columns agree, and the table is otherwise in ascending hex order — `49H/4EH` would sit exactly where `61H/62H` are printed, so the *hex* is the likelier typo. That is inference from the ordering, not a reading, and **which byte the board decodes is not settled by this manual**; treat `ESC I` / `ESC N` as the intent and verify against the board.

Note what the * marks mean: `(`, `)`, `E`, `G`, `R`, `S` and `T` are *not* board-unique — they are codes a standard system already reacts to. The manual's closing warning about them is the detection story, and it is covered in "How software finds the board" below.

The whole Osmosis product line, from their own advertising

The Portable Companion #11 (October 1983), PDF p. 32 = printed p. 30, is a full-page OSMOSIS advertisement — "The Next Step Forward" — and it gives the boards **product numbers**, which the manual we hold does not:

Product	What it is	Price
OSMOS 1	Double Density, “184k per drive using existing drives”, “fully compatible with all Osborne software” — “Install it yourself and save $165 on Osborne’s charge”	$195
OSMOS 1.5	single-sided double density now on existing drives, upgradable to 370k per drive later with new drives	$335
OSMOS 2	double-sided double density, 370k per drive	$895
OSMOS 3	double-sided double density **double tracking**, 750k per drive, “an incredible 1.5MB of portability”	$1050
OSMOS 4	disk formatting and conversion — “read, write and format **over 20 other manufactures CP/M formats**”	$225
OSMOS 5	**the internal 80-column board** — this section’s subject	$250
OSMOS 6	telex: prepare telexes in WordStar, timed send, “automatic transmission/reception in background while running any standard CP/M software”, log and queue	$1195
OSMOS 7	Winchester disks, 5¼ / 10½ / 15¾ / 21 MByte units, “a 42 Megabyte max capacity”	$2475–$3650
OSMOS 8	instrumentation — analog/digital, digital/analog, digital/digital cards	—
OSMOS 10	Olivetti ET121 to Osborne RS-232 KSR interface, 2K print buffer	$500

with a bundle — "**OSMOS 1, 4 AND 5 ALL FOR $385**" against $670 separately, "offer extended by popular demand until December 31, 1983" — and a house claim that matters for the interposer work:

> All OSMOSIS upgrades are **plug-in. There is no soldering or cutting.** Our products are supplied as kits for existing units and require no modifications to the Osborne computer. No specialized knowledge is required.

That is the design philosophy this project's interposer adopts, and the opposite of the SCREEN-PAC's soldered fly-leads (`2F00040` §6.12). Vendor address for the record: 430 Pacific Avenue, San Francisco, CA 94133, (415) 864-6372.

The 80-column board carries a hard-disk interface. The OSMOS 5 entry reads in full:

> An INTERNAL 80 column board with lots of additional features and NO alteration required to existing software: - Full width screen > 80 columns on internal and external video - Normal/ Inverted video · Underline · ⅔ and ½ intensity - Additional graphics characters for screen form design - **Includes a Hard Disk interface for future plug-in expansion** - At a very low price of only $250

So the video board was also the doorway to a hard disk — "**future** plug-in expansion", i.e. the interface ships before OSMOS 7 (the Winchester product) is something you can buy. The board this project treats as a bus-interposer template is therefore a **storage** interposer too, not only a wider screen, and that is a second reason to read its manual closely for #11/#12.

Note the attribute list as well: **normal/inverted video, underline, and ⅔ / ½ intensity**. The stock machine exposes underline and dim as attribute bits (`docs/o1-memory-io-map.md` §2a/ §3); Osmosis advertises two intensity steps. The mapping between the two is not established — relevant to #24.

What this means for this project

1. **A video personality (#24) needs four interfaces, not one.** The SCREEN-PAC's `E1A5H`/`2400H`, Nuevo's patched CBIOS, Osmosis's escape sequences and the TRA's **IEEE-port driver plus disk configuration** are four different contracts. Emulating the window is not enough; the *selection* mechanism — and, for the TRA, the *connector* — differs per board.
2. **Osmosis is prior art for the interposer, not just for video** (#11/#12): a board that takes the Z80′s socket and drives the bus is the shape being designed here.
3. **The application-patch problem is universal.** WordStar and SuperCalc needed settings on the SCREEN-PAC; dBASE needed a three-byte patch on *both* Nuevo and Osmosis. Any 80-column story for this project has to answer for software that assumes 52 columns — the practical reason the O1′s upgrades shipped with application notes.
4. **The 128-character row is the common ground.** All four widen the *window*, not the memory (`docs/o1-memory-io-map.md` §2a); only the control interface differs.

How software finds the board

Three of the four answer "is the board there?" in three different ways, and only one of them is a register — the TRA's advertisement documents no detection at all.

Board	How software detects it	Status
OCC SCREEN-PAC	read `2400H` in bank 2: **bit 2 LOW** = board fitted	documented (user guide p. 13, `docs/o1-memory-io-map.md` §2a)
Osmosis	**no detection register** — the interface is commands, so a program probes by *sending* an OS/5-unique escape and watching the screen	the manual's own warning *is* the answer; its paid Technical Manual is where "the CP/M interface" is documented
Nuevo	BIOS 1.5 **reads a port the Osborne 1 does not decode**; if it is not zero (i.e. no board), the 80-column configuration is skipped	read out of the two CBIOS images (#173); the one step needing a board is named below

Osmosis: the probe *is* the interface

The closing note to Appendix E is the answer, stated as a warning:

> Commands which are unique to the **OS/5 80-column OSBORNE 1** are annotated with an * symbol. Use of these commands on standard systems may cause spurious data to be displayed on the screen; **the program should not be otherwise affected.**

That is the contract. There is no flag to read; a program that wants to know whether the board is there sends a *-marked code and looks at the screen — and the manual is telling the programmer that doing so on a stock machine is *safe but visible*, which is a design decision rather than an omission. It is the opposite of the SCREEN-PAC's passive status bit, and it means a video personality (#24) has to model **command effect**, not a register, to be Osmosis-compatible.

Nuevo: the check is a read of an undecoded port (#173)

The DD manual says BIOS 1.5 does the check (quoted in `docs/dd-upgrade-formats.md`), and we hold CBIOS 1.5 in two forms: the DDT objects the installer inserts (`BIOS15.OBJ`, `BIOS15BM.OBJ`) and the relocated system the disk actually boots — the system tracks of `OS1NUEVO.IMD`. The latter confirms the manual's item 1: its sign-on reads

```
Osborne 1 Computer
CP/M vers 2.2
CBIOS vers 1.5
```

against the factory disk's `Osborne Computer System` / **`59k`** `CP/M vers 2.2`. The `59k` is gone, exactly as the 1.4⟶1.5 list says.

The check is the first thing the cold boot does that 1.4 does not. Both CBIOSes open identically — stack, one `CALL`, five variable loads — and then part:

	1.4 (`O1CPM.IMD`)	1.5 (`OS1NUEVO.IMD`)
cold boot	`E4ED`–`E513`, **38 bytes**	`E469`–`E4BE`, **85 bytes**
after the five variable loads	`XOR A` / `LD (0004H),A` / `INC A`	**47 new bytes**, then those same three

and those 47 bytes open with a guard:

```
E488  IN   A,(04H)      ; read port 04H
E48A  AND  1FH
E48C  JR   NZ,E4B7H     ; not present -> skip the configuration
```

Port `04H` is not a device on the Osborne 1. Its I/O decode uses only `A0`/`A1` (`docs/o1-memory-io-map.md` §4), so a *write* to `04H` is an alias of the bank-2 latch — but a **read** of it is answered by nothing on the machine, and a floating TTL bus reads back non-zero. On a stock machine the guard therefore takes the skip; with the board fitted, the board has only to pull those five bits low. That is the manual's behaviour exactly: *"If not present, the configuration routine for the 80 Column Upgrade will be skipped."*

What the guard protects is a **video** configuration, and that is the rest of the evidence:

- it reads the config byte at BIOS variable **`E1A6H`** — the address immediately above `E1A5H`, the width variable the SCREEN-PAC user guide documents (`docs/o1-memory-io-map.md` §2a), so Nuevo kept OCC's variable layout rather than inventing one;
- it then drives the **BIT 9 latch** with `OUT (06H),A` — `06H` aliases `02H`, the latch that maps bank 3, the video/dim bank, into `F000`–`FFFF` (`docs/o1-memory-io-map.md` §4). **The factory BIOS never touches that latch**: its only bank selects are ports `00H` and `01H`.

The consistency check that lifts this above a guess: BIOS 1.5 ships on **both** the DD-only disk and the 80-column disk — the installer inserts the *same* `BIOS15.OBJ` either way — and it is precisely a *shared* BIOS that needs the check, because the DD customer has no video board for the configuration routine to talk to. That is the manual's stated reason, verbatim.

The one step that needs a board: that `IN A,(04H)` reads zero with the card fitted. Everything above is read out of the two images. If that read is *not* zero with the card fitted, the guard is something else and this subsection is wrong — so it is recorded as the evidence supports it, not as established fact.

Open

- **Nuevo's presence check** — the mechanism is read out of the images (#173); the remaining step is one measurement, `IN A,(04H)` with the board fitted.

- Osmosis's *technical* manual ("theory of operation; documentation on the CP/M interface; cursor addressing") is a paid document the archive does not hold. If a detection register exists, that is where it is — but the user manual's own warning suggests there is not one.
- The DD halves of both vendors are **covered**, in `docs/dd-upgrade-formats.md`: Nuevo's boot-time auto-configuration via `NEWSETUP.COM` and its 2 x 2 / 4 x 4 formats, and Osmosis's OSMOS 1–3 density line plus the standalone OSMOS 4 conversion product.

Sources

- `research/screenpac/o1screenpacusersguide.pdf` — SCREEN-PAC User Guide, p. 13.
- `research/nuevo/OS1NUEVO.IMD` and `Nuevo_Electronics_DD_Upgrade_Manual.pdf`. Scripts extracted with `cpmcp -f osborne1`; cpmtools ships an `osborne1` diskdef matching the image exactly (`seclen 1024, sectrk 5, blocksize 1024, maxdir 64, boottrk 3`).
- `AUTOST.COM` / `AUTOST.BAK` on the same image — the 80-column feature list (extracted with `cpmcp -T raw -f osborne1`; the identical text is visible with `strings`).
- `research/osmosis/Osmosis_80_Column_Board.pdf` (+ `.ocr.txt`), §3, §4 and **Appendix E** (the escape sequences — hand-transcribed from the PDF, not the OCR).
- `research/shipped-software/O1CPM.IMD` — the factory DD boot disk used as the CBIOS 1.4 diff partner; its system tracks are the first three tracks (`boottrk 3`), converted with `disk-analyse`.
- `BIOS15.OBJ` / `BIOS15BM.OBJ` on `OS1NUEVO.IMD` — the installer's DDT objects, disassembled with `z80dasm -g 0x1F00` / `-g 0x29E0`.
- `research/technical-articles/TPC-Issue05-1983-pp98-tra-80-100-column-video-and-winchester-ads.pdf` — the TRA "80 and 100 column video" advertisement (video), and the Helix Winchester advertisement on the same page.
- `docs/o1-rom-variants.md` — ROM inventory, char-gen variants, Nuevo 1.51.
- `docs/o1-memory-io-map.md` §2a — the window, the widths, the bit fields.
- `docs/dd-upgrade-formats.md` — the DD halves of both vendors, and the Nuevo BIOS 1.4⟶1.5 list.

Software

The software library

(not yet written)

Boot disks

(not yet written)

The project

Emulation and testing

Status: Solved. Every machine boots what it should and the 3 × 3 disk matrix is measured; what is left is named in §7 — the two differing bytes in the `-blank.imd` boot area are unidentified, ROM 1.3 is undumped, and the Nuevo CBIOS 1.5 hang is recorded rather than chased.

The answer. MAME models three Osborne 1 variants and this project patches in a fourth. One `osborne1.zip` serves all three stock machines; the only ROMs absent are the four BIOS revisions that have never been dumped anywhere (§3). **All three `-blank.imd` system disks boot on all three machines** — the `52-`/`80-`/`104-` prefix is a label, not a gate (§4). The one image that does not boot is the Nuevo 80-column system disk, `research/nuevo/OS1NUEVO.IMD`, and that is closed as not-needed (#79).

Question	Answer	Where
Which machines are there?	`osborne1` (52 columns), `osborne1sp` (104), `osborne1nv` (80), plus this project's patched `osborne1nx` (#191)	§2
Which ROMs does each need?	one `osborne1.zip` serves all three; only the four never-dumped BIOS revisions are absent	§3
Which floppy images boot?	all three `-blank.imd` on all three machines; `OS1NUEVO.IMD` on none	§4
How is it run, and how is it tested headless?	windowed by hand; the Lua harness otherwise	§5, §6

The install/run mechanics are adapted from the sibling project `Myrcurial/o1prsnt` — its `o1prsnt/docs/emulating-the-osborne-1.md` and `o1prsnt/utilities/headless-testing-harness/` — which is where the macOS specifics and the VRAM-dumping technique were worked out. This document adds the per-machine ROM requirements and the boot-disk compatibility results, both measured here. (Paths prefixed `o1prsnt/` are in that repository, not this one.)

1. Install

```
brew install mame
brew install cpmtools     # cpmcp / cpmrm / cpmls - file-level disk work
```

`disk-analyse` (Keir Fraser's Disk-Utilities) is needed to convert between the IMD images MAME wants and the HFE images a hardware floppy emulator wants. It is not packaged; build it:

```
git clone https://github.com/keirf/disk-utilities.git
cd disk-utilities && make clean && make && sudo make install
```

`samdisk` (TD0 ⟶ EDSK) is **not** currently installed on this machine and is needed for the TeleDisk images in `research/` (Osmosis, RT-60A, the Zorba/Kaypro CoPower-88 sets).

2. The machines — three in MAME, and this project's fourth

MAME models three Osborne 1 variants; this project patches in a fourth. Two of MAME's three are clones of the stock machine; one has its own ROMs:

MAME machine	Description	Year	Parent	Relationship	Display (measured)
`osborne1`	Osborne-1	1981	—	the stock machine	416×240 = **52 columns**
`osborne1sp`	Osborne-1 with SCREEN-PAC	1983	`osborne1`	clone — **no ROM differences at all**	832×240 = **104 columns**
`osborne1nv`	Osborne-1 (Nuevo Video)	1984	`osborne1`	own BIOS **and** own character generator	640×240 = **80 columns**

This project adds a **fourth** (`tools/emulator-setup/mame-osborne1nx/`, #191): `osborne1nx` is a clone of `osborne1` whose Z80 sits behind an emulated card that decodes the full 8-bit I/O address, so the MD-10′s ACB-4000 host adapter can answer on ports `30h`–`33h`. It is not in MAME; it is a patch, and it runs the **1.44 ROM only**. Banking is identical to `osborne1` — measured, not assumed — and the presence check the MD-10 driver demands passes (`docs/osborne1nx-machine.md`).

Its video is not. `osborne1nx` presents the interposer's video personality, which is the **SCREEN-PAC** (#24), so its driver derives from `osborne1sp`: it answers `2400h`'s presence bit as *fitted*, takes the board's register there, and boots in the **104-column** window: `832x240`, twice the stock machine's width, with every dot of every character drawn — the same model as MAME's `osborne1sp`. Keeping the stock width by dropping every other dot column was tried and rejected: half a character's pixels makes the font unreadable (`docs/osborne1nx-machine.md` §8).

The Nuevo machine is 80 columns and the SCREEN-PAC is 104. The display width is a property of the emulated *machine*, not of the disk: it was measured by booting the same utility disk on each of the three and reading the screen device's dimensions (`tools/research/mame-boot-probe.sh`). Screenshots, all booted from `tools/emulator-setup/floppies/`:

Screenshot	Machine	Pixels	Columns
`docs/mame-screens/osborne1-52col.png`	`osborne1`	416×240	52
`docs/mame-screens/osborne1nv-80col.png`	`osborne1nv`	640×240	80
`docs/mame-screens/osborne1sp-104col.png`	`osborne1sp`	832×240	104

That is also why the disk prefix does not matter for booting (§4): the same image displays 52 columns on stock hardware and 80 on Nuevo, because the video window is the machine's, while the *text layout* is the CBIOS's.

The SCREEN-PAC is modelled as *hardware only* (the 80/104-column logic lives on the board, not in a ROM), whereas Nuevo shipped a replacement monitor ROM. That is independent evidence that the two 80-column upgrades are different designs — see ##73 and `docs/o1-rom-variants.md`.

The SCREEN-PAC has three modes, and MAME models one of them

The measured 104 columns is the SCREEN-PAC's *default*, not its only width. Its own user guide gives the full set, and all of them are selectable **by the stock BIOS**:

> SCREEN-PAC has preset video contrast in **80 and 104 column format** … In these same formats, SCREEN-PAC provides **horizontal scrolling by 32 column blocks** … At BIOS location **E1A5H**, POKE or store **0 for 52 columns / 1 for 104 columns / 3 for 80 columns**. CALL BIOS location **E4E5H**. — *Osborne 1 SCREEN-PAC User Guide*, p. 13 (PDF p. 17)

So 52 / 80 / 104 are three widths of **one window** (`o1-memory-io-map.md` §2a), and the stock ROM already contains the routine that switches between them — which is why "no ROM differences at all" is the right description of `osborne1sp`. MAME models the 104-column window; the **80-column mode is selected by poking `E1A5H` = 3 and calling `E4E5H`** — read from the user guide, and the poke itself is not yet measured here.

Presence flag. The same page records how software detects the upgrade with no ROM change:

> reading 2400H (Bank 2) returns data **bit 2 LOW**. Without SCREEN-PAC, this bit is read HIGH. This feature allows different screen responses and menus depending on whether the Osborne is fitted with SCREEN-PAC.

`2400H` in bank 2 sits inside the memory-mapped I/O window and is recorded there as a **read-only presence bit, not a register** (`o1-memory-io-map.md` §3).

3. ROM requirements

Every ROM any of the three machines needs, with the CRC32 MAME asserts against. Full inventory and provenance: `docs/o1-rom-variants.md`.

ROM file	Size	CRC32	`osborne1`	`osborne1sp`	`osborne1nv`	Role
`rev1.40.ud11`	4096	`3d966335`	✓ (BIOS 1.4)	✓	—	monitor ROM 1.4
`rev1.43.ud11`	4096	`91a48e3c`	✓ (BIOS 1.43)	✓	—	monitor ROM 1.43
`3a10082-00rev-e.ud11`	4096	`c0596b14`	✓ (BIOS 1.44)	✓	—	monitor ROM 1.44 — **the default**
`char.ua15`	2048	`5297c109`	✓ (BIOS ≤ 1.4)	✓	—	early character generator
`7a3007-00.ud15`	2048	`6c1eab0d`	✓ (BIOS 1.43/1.44)	✓	✓	late character generator
`monrom-rev1.51-12.ud14`	4096	`298da402`	—	—	✓	Nuevo monitor ROM 1.51
`character_generator_6-29-84.14`	2048	`6c1eab0d`	—	—	✓	Nuevo-era char-gen — **same bytes** as `7a3007-00.ud15`
`osba.bin`, `osb12.bin`, `osb121.bin`, `osb13.bin`	4096	—	NO_DUMP	NO_DUMP	—	BIOS A / 1.2 / 1.2.1 / 1.3 — **never dumped**

One `osborne1.zip` serves all three machines. The repository's `tools/emulator-setup/roms/osborne1.zip` holds the six available files, and MAME's hash fallback satisfies `character_generator_6-29-84.14` from `7a3007-00.ud15` because the bytes are identical. Verified:

```
$ mame osborne1nv -verifyroms -rompath tools/emulator-setup/roms
romset osborne1nv [osborne1] is good
$ mame osborne1 -verifyroms -rompath tools/emulator-setup/roms
osborne1    : osba.bin (4096 bytes) - NOT FOUND - NO GOOD DUMP KNOWN
... (the four undumped BIOSes only)
romset osborne1 is best available
```

"best available" is the correct and expected result for `osborne1`/`osborne1sp`: the only missing ROMs are the four that have never been dumped anywhere. The default BIOS is **1.44** (`3a10082-00rev-e.ud11`), which is what the banner shows.

Where to get them

The upstream collection used by `o1prsnt` is archive.org/details/osborne-1-roms (two files: `occ-7a3007-00-reva.rom`, `occ-v1.44.rom` — enough for `osborne1` and `osborne1sp`, **not** enough for `osborne1nv`). This repository carries the full set in `tools/emulator-setup/roms/`, archived with provenance in `research/roms/`.

Rompath layouts

MAME looks for machine ROMs in `<rompath>/<machine>.zip` or `<rompath>/<machine>/`, **and it will also fall back to a loose file in the rompath root** when a required ROM is missing but a file with the right CRC is present. Measured, both ways:

```
$ mame osborne1nv -verifyroms -rompath tools/emulator-setup/roms
romset osborne1nv [osborne1] is good          # loose 7a3007-00.ud15 satisfies
                                              # character_generator_6-29-84.14 by
CRC
$ # the same files in a plain <rompath>/osborne1/ directory:
osborne1nv  : character_generator_6-29-84.14 (2048 bytes) - NOT FOUND (osborne1)
```

So the CRC fallback works for **loose-root files and for files inside a zip**, but **not** for files inside a romset *directory*. The wrapper's temporary `<rompath>/osborne1` symlink (used when the rompath has no `osborne1.zip`) turns a working loose rompath into a failing directory — which is why `tools/emulator-setup/roms` keeps `osborne1.zip` alongside the loose files. This matters when substituting a third-party ROM: build a **zip**, see §5.

4. Boot disks: what actually boots on which machine

Measured, not assumed. Each cell is a headless run of the harness in §6 that presses RETURN at the BIOS prompt and waits for an `A>`.

Image	osborne1	osborne1sp	osborne1nv
tools/emulator-setup/floppies/52-blank.imd	**BOOTED**	**BOOTED**	**BOOTED**
tools/emulator-setup/floppies/80-blank.imd	**BOOTED**	**BOOTED**	**BOOTED**
tools/emulator-setup/floppies/104-blank.imd	**BOOTED**	**BOOTED**	**BOOTED**
research/nuevo/OS1NUEVO.IMD	no-boot	—	no-boot
research/nuevo/OS1NUEVO.IMD (pristine upstream copy)	—	—	no-boot

The `52-`/`80-`/`104-` prefix is a label, not a gate

All three images boot on all three drivers, because all three are the *same* CP/M system disk — they differ by exactly **two bytes**:

```
$ cmp -l 52-blank.img 80-blank.img
  5737 377   0        # 52 = 0xFF, 80 = 0x00
  5798   0   3        # 52 = 0x00, 80 = 0x03
$ cmp -l 52-blank.img 104-blank.img
  5737 377   0        # 104 = 0x00
  5798   0   1        # 104 = 0x01
```

Both offsets are in the boot area (track 1, sector 1), which is where a CBIOS would keep a video-mode selection. Whatever those two bytes select, they do not change which machines the disk boots on.

So what *does* gate bootability is the **CBIOS on the disk**, not the filename prefix. The prefix is a useful human label — keep using it — but do not rely on it to decide the driver. `tools/check_o1_map.sh` still derives the driver from it (`52-` → `osborne1`, `104-` → `osborne1sp`, `80-` → `osborne1nv`), warns when it selects anything but the stock machine, and takes `O1_MACHINE` as an override; `tools/README.md` states the same rule. That inference is §7 item 1.

What the `-blank.imd` disks actually are

Despite the name, they are **not blank** — they are Osborne 1 double-density CP/M 2.2 system disks, IBM-MFM, 40 tracks × 5 × 1024 bytes, single-sided:

```
autost.com  cbas2.com  copy.com  crun2.com  mbasic.com  movcpm.com
pip.com  setup.com  sysgen.com  xdir.com  xref.com
```

The system area identifies them as `Osborne Computer System / 59k CP/M vers 2.2 / CBIOS 1.4`, with `SETUP VERS. 2.6` and `PIP VERS 1.5`. `AUTOST.COM` runs XDIR (the `Extended Directory version 3.5` banner) rather than booting to a bare prompt — the same behaviour `o1prsnt`'s `DISCOVERY.md` records.

`research/nuevo/OS1NUEVO.IMD` — the Nuevo 80-column system disk

This *is* the Nuevo boot disk. Its label is `CP/M 2.2 system disk for Osborne 1 w/ Nuevo DD & 80 column mods`, and its system area carries a CBIOS that is explicitly the 80-column one:

```
Osborne 1 Computer / CP/M vers 2.2 / CBIOS vers 1.5
*  THIS IS AN EIGHTY (80) COLUMN DISPLAY FOR THE OSBORNE ONE.  *
*  (3) FULL SCREEN REVERSE VIDEO CAPABILITY (JUMPER SELECT).
*  (4) CURSOR PROGRAMMABLE AS UNDERLINE OR REVERSE VIDEO BLOCK.
*  (6) REVERSE VIDEO BLOCK CAPABILITY (INSTEAD OF UNDERLINE; JUMPER SELECT).
```

It does not boot under MAME. On `osborne1nv` it loads, prints the CBIOS 1.5 banner, and then sits there — screen unchanged over 260 emulated seconds, with RETURN and space nudges. On the stock `osborne1` it does the same.

That is not a bad copy: the pristine upstream file from Dave Dunfield's `osborne1.zip` behaves identically, and our copy differs from it only in the IMD comment block. So either MAME's `osborne1nv` does not reproduce whatever the Nuevo CBIOS waits on, or the disk needs the physical Nuevo card to get past initialisation. **Closed as not-needed — see #79.**

Decision: the Nuevo path is not pursued

The video personality (#24) does **not** depend on answering the above. The decision taken is to build it from the **ScreenPac design plus OZROM's soft 52/80/104-column switch**, and to treat the Nuevo board as prior art (#73) rather than as a target. So the Nuevo CBIOS hang is not on the critical path, and ##79 is closed with the question recorded rather than chased — nothing in the Nuevo design has yet turned up a feature worth the validation cost.

Practical answer for Nuevo work

- To get a **prompt** on `osborne1nv`, use any of the three `-blank.imd` disks.
- To exercise the **Nuevo 80-column video path**, `OS1NUEVO.IMD` is the right disk but it cannot get past CBIOS initialisation; per the decision above this is not a blocker for #24.
- Material for *building* a working Nuevo system disk, if anyone ever wants it: `OS1NUEVO.IMD` carries the Nuevo CBIOS 1.5 in its system tracks, the `-blank.imd` disks carry `SYSGEN.COM` (and `MOVCPM.COM`) for writing a new system, and `research/nuevo/` has the Nuevo DD upgrade manual and the Nuevo BIOS ROM (`monrom-rev1.51-12.ud11`). A custom build is therefore plausible.

5. Running it

Directory layout

`o1prsnt` settled on a stable layout so that launching MAME from odd places does not scatter `cfg/` and `snap/` directories everywhere. Our tooling defaults to the same paths, so both projects share one tree:

```
~/Documents/Osborne1/
  roms/           osborne1.zip
  floppies/       .imd / .hfe disk images
  o1prsnt/        working copies
  presentations/  slide sources
```

`tools/check_o1_map.sh` and `tools/emulator-setup/` read `~/Documents/Osborne1/roms` and `~/Documents/Osborne1/floppies` by default; override with `O1_ROMPATH` and `O1_FLOPPY`.

Windowed launch

```
cd ~/Documents/Osborne1
mame osborne1   -flop1 ~/Documents/Osborne1/floppies/52-blank.imd
mame osborne1sp -flop1 ~/Documents/Osborne1/floppies/104-blank.imd
mame osborne1nv -flop1 ~/Documents/Osborne1/floppies/80-blank.imd
```

The machine name is explicit — do not let the image name imply it (§4).

macOS: MAME forces fullscreen. LEFT Option + Return toggles out of it, and Cmd-Q is not honoured — use the red window button.

File-level disk work (`cpmtools`)

```
cpmls -f osborne1 IMAGE                          # list
cpmcp -f osborne1 IMAGE host.txt 0:CONTENT.TXT   # copy in
cpmrm -f osborne1 IMAGE 0:CONTENT.TXT            # remove
```

The `osborne1` disk definition matches the double-density format these images use (40 × 5 × 1024).

IMD ↔ HFE

For a hardware floppy emulator:

```
disk-analyse in.imd out.hfe
disk-analyse in.hfe out.imd
```

Running with a third-party ROM (OZROM 1E)

MAME cannot be told about a BIOS that is not in its ROM definitions, but it *will* load a file whose bytes do not match the expected hash — it warns and carries on. So a third-party monitor ROM is run by presenting it under one of the stock BIOS filenames. Build a **romset zip**, not a plain directory:

```
mkdir -p /tmp/ozbuild/osborne1 && cd /tmp/ozbuild/osborne1
cp <repo>/tools/emulator-setup/roms/*.ud11 \
   <repo>/tools/emulator-setup/roms/*.ud15 \
   <repo>/tools/emulator-setup/roms/*.ua15 .
cp <repo>/research/roms/OZROM-1E/OZROM_1E.BIN 3a10082-00rev-e.ud11
cp 7a3007-00.ud15 character_generator_6-29-84.14    # see note below
mkdir -p /tmp/ozset && zip -q -X /tmp/ozset/osborne1.zip *

O1_ROMPATH=/tmp/ozset tools/research/mame-boot-probe.sh \
    osborne1 tools/emulator-setup/floppies/52-blank.imd 120 /tmp/ozrom.png
```

Why a zip and not `mkdir .../osborne1`. The simpler directory recipe boots `osborne1` and `osborne1sp`, but **fails on `osborne1nv`**: that machine needs `character_generator_6-29-84.14`, which MAME satisfies by CRC fallback from `7a3007-00.ud15` only when the files are loose in the rompath root or inside a zip — *not* when they sit in an `osborne1/` directory. Copying the chargen under its own name (it is byte-identical, `o1-rom-variants.md`) makes the fallback deterministic for all three. Verified: `mame osborne1nv -verifyroms -rompath /tmp/ozset` reports *good*.

OZROM 1E boots and runs on all three machines. Measured:

```
osborne1    416x240  (52 columns)  ->  A>
osborne1sp  832x240  (104 columns) ->  A>
osborne1nv  640x240  (80 columns)  ->  A>
banner: OZROM 1E #00187 / (c) 1984 Micro Management
```

MAME prints *"the machine might not run correctly"* because the ROM hash is not the one it expects — that warning is the point, not a problem. **The full OZROM treatment — memory map, ROM jump table, keytable, the column switch and its mechanism, clock, memory tester — is `docs/ozrom-1e.md` (#81).**

Why this matters to us. OZROM 1E is not a ROM swap with a nicer font; per its manual (`research/roms/OZROM-1E/OZROM_1E_Manual.pdf`, Micro Management, Manual Rev. A 84/09/01) it adds capability we would otherwise have to build:

- **User-selectable 52 / 80 / 104-column display from the keyboard** — a documented *software* video-mode switch. Directly relevant to the video personalities (#24) and to how the Nuevo and ScreenPac paths are driven. On a machine with no 80-column upgrade it can also toggle between the left and right 52 columns of an 80-column display.
- **A system memory tester** that reports which chip to replace — a validation tool for #29.
- **A software-redefinable keyboard** — individual keys at any time, plus eight locally-redefinable function keys of up to 63 characters each — relevant to #45/#47.
- **Time-of-day clock support** (`TIME`, `ALARM`, `TIMER`, and a way to set the clock) — relevant to #7/#18.
- **Documented interfaces**: an interrupt hook for programmers, the ROM jump table, the OZROM memory map and keytable, plus notes on 1793/1797 floppy controller operation (#20) and truer Televideo 912/920 emulation.
- Removes the "Centronics printer not connected" lockout on double-density systems (#19).

What it gives up, stated precisely in its own Appendix C (the manual is now OCR'd to `research/roms/OZROM-1E/OZROM_1E_Manual.ocr.txt`):

> **IEEE-488 functions.** The OZROM 1E does not support the IEEE-488 functions. Although Centronics printer operation is not affected, some printers … and some data acquisition and analysis devices … require the IEEE-488 routines to operate. In this instance, extra software would be required …

The jump-table appendix is blunt about it: the entries at `013F–0156` *were* the IEEE-488 routines and **calling them now produces a controlled system crash**, and *"IEEE-488 AS IOBYTE DEVICE 3 NOW NULL DEVICE"* — the status entries return `0FFH`, input returns zero, output does nothing. The parallel/Centronics entries at `0193+` survive.

So OZROM and Drive C are mutually exclusive. Drive C is an IEEE-488 device and OZROM removes the routines it talks through; the manual's "extra software would be required" is exactly the driver we have the source for in `rom144.asm` — so it is recoverable, but it is not free.

One more difference worth knowing, because it constrains any 2.0 ROM (#83): OZROM **shrinks the video window to 3 KB** (`F000–FBFF`) and uses the last 1 KB (`FC00–FFFF`) for its redefinable keyboard and function keys. It therefore scrolls by moving the 23 lines above rather than by moving the window. The stock ROM's 4 KB video RAM map in `docs/o1-memory-io-map.md` §2 does not apply to OZROM.

6. Headless testing

The technique is `o1prsnt`'s (`utilities/headless-testing-harness/README.md` for the API, `DISCOVERY.md` for how it was derived):

```
SDL_VIDEODRIVER=dummy mame MACHINE -flop1 IMAGE -video none -sound none \
  -nothrottle -autoboot_script RUN.lua -seconds_to_run N -rompath ROMS
```

- `SDL_VIDEODRIVER=dummy` is **mandatory** from a pure terminal session on macOS; without it MAME aborts with "Could not initialize SDL".
- `-video none -sound none` — no *renderer*, no audio. The emulated screen still renders into VRAM, which is all a test reads.
  - **`-video none` is not "no window"; `SDL_VIDEODRIVER=dummy` is what makes a run headless, and it only works for an SDL build.** The window belongs to the OSD, and MAME's default is **full screen** (`-window`: *"enable window mode; otherwise, full screen mode is assumed"*). The **mac OSD** has no `SDL_VIDEODRIVER` to hide behind, so a run there takes the whole display with a grey window until it exits — `-window` is the only thing that stops it, and `-numscreens 0` is rejected (`must be between 1 and 4`). The fix is not to add `-window` everywhere: it is to build with the SDL OSD, which is the tree's default on macOS, and that needs **`pkg-config`** so MAME can find `sdl3` (`REQUIREMENTS.md`). A source build without pkg-config cannot find `<SDL3/SDL.h>` and falls back to the mac OSD — `apply.sh` says so out loud when it does. Measured on the `osborne1nx` build, 2026-10-12.
- The screen is read straight from video RAM: base `0xF000`, **128 bytes per row**, 32 rows. Character codes are readable directly; the 9th (attribute) bit lives in a separate bank and can be ignored for text.
- `-nothrottle` runs the 4 MHz Z80 as fast as the host allows (~20–25×) but **`emu.wait()` and `-seconds_to_run` count emulated seconds**, so any timing you measure is a true 4 MHz Z80 timing and transfers to hardware.
- End the script with `os.exit(0)`; letting MAME hit the budget often ends in a harmless but noisy macOS teardown segfault.

The two harnesses in this family

Harness	Where	Purpose
`tools/o1_mame_probe.lua` + `tools/check_o1_map.sh`	this repo	asserts `docs/o1-memory-io-map.md` (12 checks) against the running machine
`tools/research/mame-boot-probe.{lua,sh}`	this repo	boot one image on one machine: `A>` or not, screen dump, **screen geometry**, optional PNG
`o1harness.lua` + `run-test.sh`	`o1prsnt`	keyboard injection (`H.boot_cpm()`, `H.type_line()`) and screen assertions (`H.wait_for()`, `H.find()`, `H.dump()`)

The boot probe is the one that produced the geometry table in §2 and the screenshots in `docs/mame-screens/`. It is deliberately dependency-free; for scripted tests that need to *type* at the machine, use `o1prsnt`'s harness.

Screenshots. MAME's Lua screen device has `screen:snapshot(path)`, which works headless (`-video none`) and needs no display:

```
manager.machine.screens[":screen"]:snapshot("/tmp/shot.png")
```

The §4 boot matrix was produced with the second one plus a ten-line script:

```
cat o1harness.lua probe.lua > /tmp/run.lua
SDL_VIDEODRIVER=dummy mame osborne1nv -flop1 IMAGE -video none -sound none \
  -nothrottle -autoboot_script /tmp/run.lua -seconds_to_run 120 \
  -rompath tools/emulator-setup/roms
```

Gotchas worth not rediscovering

1. **VRAM is a ring buffer.** The machine scrolls by rotating a hardware start register, not by moving memory, so a given screen line moves between VRAM rows after any scroll. Assert with whole-buffer substring searches, never fixed row reads.
2. **Pace your typing.** `natkeyboard:post()` drops characters if a second post arrives while the first is still being typed, and the guest flushes its input buffer around disk I/O. Wait for the prompt, then type (~0.15 s/character).
3. **MBASIC's `Ok` prompts are stale-prone** — count them (`H.ok_count()`) rather than matching `"Ok"`.
4. **MAME writes `cfg/` and `snap/` into the current directory** — run from a neutral directory so neither the repo nor your home dir collects litter.
5. The BIOS **waits for RETURN** ("Insert disk in Drive A and press RETURN.") before it loads anything; every automated test must press it.

7. Open items

Open:

1. **The prefix → driver inference** in `tools/check_o1_map.sh` needs revisiting (§4): the `52-`/`80-`/`104-` prefix does not determine what a disk can boot.
2. **Identify the two differing bytes** at offsets 5737 and 5798 in the `-blank.imd` set — presumably a CBIOS video-mode selector, but unverified. A small disassembly job against the CBIOS in the system tracks.
3. **ROM 1.3 is undumped** and we hold its source disk — see `docs/o1-rom-variants.md`.
4. **Driving the OZROM column keys from the harness** — the ctrl-shift 52/80/104-column select keys cannot be posted, because the harness types plain characters. Routed to **#167**; wanted by #24 as much as by #81 (`docs/ozrom-1e.md` §9).

Closed, and why:

- **#79** — why the Nuevo CBIOS 1.5 hangs (`OS1NUEVO.IMD`). Closed as not needed: the Nuevo path is not pursued (§4) and #24 is built from the ScreenPac design plus OZROM's column switch. The hang is recorded rather than chased — and it is not a bad copy: Dunfield's pristine upstream behaves identically.
- **OZROM 1E as a tool (#81)** — closed. It boots on all three machines (§5) and its manual is indexed in **`docs/ozrom-1e.md`**: memory map, ROM jump table, keytable, the 52/80/104-column switch and its mechanism, the clock and the memory tester.

8. Sources

- Adaption base: `Myrcurial/o1prsnt` — its `docs/emulating-the-osborne-1.md`, `utilities/headless-testing-harness/{README,DISCOVERY}.md` and `scripts/validate-emulator.sh`.
- Machine/ROM structure: MAME `src/mame/osborne/osborne1.cpp` (`ROM_START( osborne1 )`, `ROM_START( osborne1nv )`, the `COMP()` lines).
- ROM inventory and CRCs: `docs/o1-rom-variants.md`.
- The memory and I/O map the probes assert: `docs/o1-memory-io-map.md` (§2a for the video geometry, §3 for the memory-mapped I/O window).
- OZROM 1E's manual, ROM and interfaces: `docs/ozrom-1e.md` (#81).
- The fourth machine: `docs/osborne1nx-machine.md` (#191).
- Archive provenance: `research/README.md`.
- Disk pipeline and machine checks: `tools/README.md`.

The interposer and its virtual peripherals

Status: Solved as a design. The bus position, the three taps, the banked ROM window, the memory window and the port-base strategy are settled, and the parts of them that touch the machine are measured; §7′s personality *selection* is still a proposal, and nothing here is built.

How the RP2354B interposer sits on the Z80 bus and where each virtual peripheral lives in the Osborne 1 memory and I/O maps.

Serves #12 (the electrical/mechanical interface spec) and #16 (the personalities that live in the address space this allocates).

1. Where the interposer physically sits

The Osborne 1 exposes the Z80 on a 40-pin DIP socket. Like the CoPower-88 and the OCC ScreenPac, the interposer **plugs into the Z80 socket** and the Z80 plugs into the interposer — so it sees (and can drive) every bus cycle.

```
flowchart TB
    subgraph MB["Osborne 1 mainboard"]
        ZSKT["40-pin Z80 socket<br/>(A0-A15, D0-D7,<br/>MREQ/IORQ/RD/WR/M1,<br/
>RFSH, HALT, INT, NMI, RESET, CLK)"]
        ROMSKT["Main ROM socket<br/>UD11, 24-pin<br/>(4KB BIOS at 0000-0FFF)"]
        RAM["64KB DRAM<br/>(banked)"]
        VRAM["Video RAM<br/>F000-FFFF window"]
        IO["Onboard I/O<br/>(FDC, PIO, SIO, KBD)"]
        CG["Char-gen ROM socket<br/>UA15, 24-pin<br/>(font data + scan lines)"]
        ZSKT --- ROMSKT & RAM & VRAM & IO
    end

    subgraph INTP["RP2354B Interposer (in Z80 socket)"]
        direction TB
        Z80["Original Z80A<br/>(NMOS, 4 MHz)<br/>reinstalled here"]
        BUF["74LVC245 buffers /<br/>level shifters<br/>(5V Z80 <-> 3.3V
RP2354B)"]
        PIO["RP2354B PIO + DMA<br/>bus sniffer / driver"]
        MCU["RP2354B cores<br/>cycle decoder +<br/>peripheral firmware"]
        WIFI["WiFi / USB<br/>stream out"]
        VID["Video renderer<br/>(shadow VRAM + font)<br/>DVI/HDMI via RP2354B
HSTX"]
        HDMI["HDMI monitor out"]
        Z80 <--> BUF <--> PIO <--> MCU --> WIFI
        PIO --> VID --> HDMI
    end

    ZSKT <-."A/D/MREQ/IORQ/RD/WR<br/>tapped + can be driven".-> BUF
    ROMSKT <-."main ROM socket tap<br/>(Pico serves the window, #83)".-> BUF
    CG <-."char-gen socket tap<br/>(ScreenPac-style, #24)".-> VID
```

Tap points: three. The interposer sits in the Z80 socket, so it already has the bus; the other two are socket lead-sets, and between them they supply the two things the bus cannot.

Tap	Socket	Gives	Serves
bus	Z80, 40-pin	every cycle, including all video-RAM writes	all personalities
ROM	**main ROM, UD11, 24-pin**	the ROM window itself, through a daughterboard (DIP down, ribbon up) — the Pico **serves** it, so no physical ROM part is needed	**ROM 2.00 (#83)**, one image, any OEM-faithful mainboard
font	char-gen, UA15, 24-pin	the font, read-only	the HDMI video personality (§5), #44

The ROM tap is what makes a banked ROM possible at all, and it is cheaper than it looks: a 24-pin 2732 socket has no `A12`, no `A13` and no write strobe, so a *part* in it could never exceed 4 KB — but when the **Pico serves the window**, the address lines past `A11` do not matter, because the bank is which 4 KB the firmware presents. No new part, no programmer, updatable in flash. A physical 27128 is only for a board with no interposer. The SFF board grafts the same capability in.

One host to design for. An existing **OEM board + interposer** is the reference, and any faithful 1:1 recreation of *just* the OEM board with the Dual Density upgrade built in would be equivalent to it — so nothing in this document needs a second host, and no second case.

Level shifting note (#11): the host Z80A is NMOS (confirmed, date 8408). NMOS outputs are TTL-ish; NMOS inputs have ~2.0–2.4 V high thresholds. Use **74AHCT/74HCT toward the Z80** (TTL-compatible inputs) and **74LVC245 host-side**. ScreenPac used raw 74HC; we prefer AHCT/LVC for margin across all three target machines.

2. Z80 memory map and where virtual devices attach

The O1 banks ROM, RAM, and video in a 64KB space. The interposer claims a small decode window by **snooping MREQ and either passing through or overriding** the onboard response.

The ROM window is claimable too. With the main-ROM socket tap (§1) the interposer holds the ROM window and can fill it, so **the window becomes a banked one** — which is how ROM 2.00 (#83) fits 11776 bytes of monitor + CCP + BDOS + CBIOS into a machine whose ROM window is only 8 KB. The window is `0000–1FFF`, and it is already 8 KB wide: the mainboard never decoded `A12`, so the stock machine reads the same 4 KB monitor twice across it, and a softROM that decodes `A12` gets two contiguous 4 KB pages — page 0 the monitor, page 1 (`1000–1FFF`) the payload — **with no latch and no register** (`docs/o1-rom-2.0.md` §4.1b, measured in the emulated machine).

The ROM's register pair with the firmware is `2800h`/`2801h`, not `2300h`. The monitor writes a request to `2800h` and polls `2801h` for the reply (`docs/rom-pico-protocol.md` §2; the residue is §7). `2300–23FF` was the first choice, and it is not free: the onboard selects check two address bits

each, so `0x2300` is the floppy's command/status register and `0x2301` is the floppy's **track register** *and* the keyboard's row 0 — which is why the door moved (`docs/o1-mainboard-schematic.md` §7a and `tools/o1_mame_probe.lua`'s C9 report, both measured). **The door's address is open issue #276** — the same repair, filed against the places in the tree where the stale pair survived. With the Pico serving the window that register is firmware, not a hardware latch. The 4 KB the video RAM does not use is not the only spare space on this machine; the ROM mirror is another.

```
flowchart LR
    subgraph MAP["Z80 64KB memory map"]
        direction TB
        R0["0000-0FFF<br/>BIOS ROM<br/>(bank 0)"]
        R1["1000-EFFF<br/>System RAM<br/>(shared both banks)"]
        R2["F000-FFFF<br/>Video RAM<br/>(bank-switched window)"]
    end

    subgraph VW["Interposer-claimed window (config)"]
        VP["Virtual-peripheral<br/>mailbox RAM<br/>(e.g. EF00-EFFF)"]
    end

    MCU2["RP2354B firmware"]
    MAP -->|"MREQ+A15-A0 snooped"| DEC{"Decode<br/>match?"}
    DEC -->|"no: pass through"| MB2["Onboard ROM/RAM<br/>respond normally"]
    DEC -->|"yes: assert override,<br/>tri-state host, drive bus"| VP
    VP <--> MCU2
```

Banking caveat

The O1 video RAM and BIOS share the top of the map via a bank-select I/O bit. Any memory-resident virtual device must **(a)** live in an address range the current bank exposes and **(b)** defer to the real video RAM when it is mapped in. Safest is a small mailbox page in the always-present RAM region, or an I/O-port device instead (below).

3. Z80 I/O map and the virtual-peripheral ports

I/O-mapped devices are cleaner: they never collide with the memory banks and are how every real O1 peripheral (and the CoPower-88) already works.

```
flowchart TB
    subgraph IOP["Z80 I/O space (A7-A0 decoded)"]
        direction TB
        ONB["Onboard ports<br/>FDC / PIO / SIO / KBD<br/>(existing decode)"]
        CP88["CoPower-88 mailbox<br/>0x7E data / 0x7F status<br/>(or
0xFE/0xFF)<br/>*real hw being reverse-eng*"]
        V1["VIRT: net disk<br/>port base B+0..B+3"]
        V2["VIRT: WiFi modem<br/>port base B+4..B+7"]
        V3["VIRT: RTC<br/>port base B+8..B+B"]
        V4["VIRT: virtual MX-80<br/>printer port base B+C..B+F"]
    end

    Z802["Z80<br/>IN/OUT (IORQ+RD/WR)"] --> DEC2{"Port decode<br/>A7-A0"}
```

```
DEC2 --> ONB
DEC2 --> CP88
DEC2 -->|"claim free base B"| V1 & V2 & V3 & V4
V1 & V2 & V3 & V4 <--> FW["RP2354B<br/>peripheral firmware"]
FW --> NET["WiFi / SD /<br/>host stream"]
```

Port-base strategy: pick a base `B` in a region the onboard decode leaves free **and** that does not collide with the CoPower-88′s 0x7E/0x7F (Kaypro) or 0xFE/0xFF (Zorba) — so the interposer can *emulate or coexist with* the real CoPower. The comparator-based decode on the real CoPower board (the "16-2103" IC, see photo-survey) is the model for a clean single-port-pair decode.

The interposer's **own** host↔firmware channel — the mailbox, the one port the machine's CP/M tooling talks to directly — is specified separately in `docs/mailbox-protocol.md` (#155): it takes a port pair of its own, chosen so that a leaked write cannot move the bank latches.

4. The virtual peripherals (from the roadmap)

Virtual peripheral	Bus type	Address/Port	Host interface	Back end
Network disk	I/O ports	base+0..3 (cfg)	command/data/status FIFO	SD image / WiFi block server
WiFi modem	I/O ports	base+4..7 (cfg)	8250-ish UART regs	TCP/Telnet over WiFi
RTC	I/O ports	base+8..B (cfg)	RT-60A-compatible regs (see #7)	RP2354B RTC / NTP
Virtual MX-80 printer	I/O ports	base+C..F (cfg)	Centronics-style data+strobe	WiFi / USB to host spooler
Video / HDMI output (ScreenPac personality, #24)	memory snoop + char-gen tap	VRAM `F000-FFFF` (passive) + UA15 socket	none — transparent to host	DVI/HDMI via RP2354B HSTX (§5)

The four I/O peripherals are each an independent decode window; firmware registers a `{base, mask, read_fn, write_fn}` entry and the cycle decoder dispatches. The video personality is different: it never answers a bus cycle at all — it only *listens* (and taps the char-gen socket), so it cannot collide with any onboard or virtual device.

5. Video output — ScreenPac personality and HDMI out (#24)

The interposer's second personality is video. The OCC ScreenPac is the template: it taps **both** the Z80 40-pin socket **and** the character generator ROM socket (UA15, 24-pin), and shadows the display through dual-port video RAM (6116s + 74HC157 muxes). The interposer reproduces that function in firmware and extends it to a modern output:

1. **VRAM snoop.** Video RAM lives at `F000-FFFF` (128×32 cells, 52×24 visible). Sitting in the Z80 socket, the interposer already sees every bus cycle; writes into the VRAM window update a shadow copy in RP2354B RAM.
2. **Font capture.** The 2716 character ROM (128 chars × 8×10 pixels) is dumped once at init through the UA15 tap — or a known-good font image is preloaded in firmware — and frames are re-rendered from shadow VRAM + font. Tapping UA15 like the ScreenPac did also tracks machines running replacement/upgrade char ROMs. The tap is **read-only by necessity**: on the mainboard `/CE` (18) and `/OE` (20) are bussed with GND and grounded, so the ROM is permanently selected and always driving `O0–O7` (`o1-mainboard-schematic.md` §4a). Reading costs nothing; *replacing* the font would mean intercepting those two pins between socket and ROM, so the design keeps the fitted ROM as the source of truth and preloads a font image for the 80/104-column modes instead.
3. **Render + output.** The second RP2354B core renders the shadow frame and drives DVI/HDMI via the RP2354B **HSTX** peripheral (or PicoDVI-style bitbang). O1 timing is gentle: 15.9744 MHz master, 8 MHz dot clock, 60 Hz frame — comfortably inside DVI 640×480 margins with integer upscale.

Prior art — Pico-based bus video cards (candidate boards to study)

- **A2DVI** — `github.com/rallepalaveev/A2DVI`: Apple II digital video card, RP2040-based, HDMI/DVI out. Closest analogue: a Pico-class MCU on a vintage bus rendering HDMI in real time.
- **V2RetroComputing "analog"** — `github.com/V2RetroComputing/analog`: Pico-based Apple II card outputting DVI video over an HDMI connector.
- **AppleII-VGA** — `github.com/markadev/AppleII-VGA`: Pico-based VGA card; same snoop-and-render architecture, VGA instead of DVI.
- **PicoDVI** — `github.com/Wren6991/PicoDVI`: the RP2040 DVI bitbang library; on the RP2354B the HSTX hardware does the heavy lifting instead.

All four prove the pattern this personality needs: a Pico-class MCU snooping a vintage bus and emitting stable modern video. Unlike the Apple II cards, the interposer sits *in the CPU socket* (ScreenPac-style), so it also sees bank-switch writes and can track which bank currently owns `F000-FFFF` (video RAM vs the dim-bit attribute bank — see `docs/o1-memory-io-map.md`).

> Relationship to the real ScreenPac: the reverse-engineering phase sniffs the real board like the other targets (#9 photos; service manual safety copy in `research/screenpac/`). The HDMI personality is then the "virtual ScreenPac" — plus a display output the original never had.

6. Coexistence with the real peripherals being reverse-engineered

```
flowchart LR
    subgraph REAL["Real hardware (reverse-eng targets)"]
        CP["CoPower-88<br/>0x7E/0x7F"]
        DC["Drive C<br/>IEEE-488 ramdisk"]
        RTC["RT-60A<br/>RTC"]
        ACT["ACT hard disk<br/>OCC1"]
```

```
    end
    subgraph VIRT["Interposer emulation"]
        VCP["virtual CoPower"]
        VDC["virtual Drive C"]
        VRTC["virtual RTC"]
        VACT["virtual HD"]
    end
    REAL -->|"sniff real cycles<br/>to learn protocol"| LEARN["RP2354B<br/>
>capture"]
    LEARN --> VIRT
    VIRT -->|"later: replace real hw<br/>on same ports"| HOST["O1 runs
unchanged<br/>software"]
```

The interposer first **passively sniffs** each real device to pin its protocol (the reverse-engineering phase), then can **emulate** it on the same decode so the original software runs against the virtual version.

7. Personality selection — the gate, and where each personality stands

No decisions have been taken here. This section is the *input* to that decision: the rule we apply, what it misses, and the resulting map — recorded so the choice is argued once rather than re-derived per personality.

The current rule, and its gap

`firmware/README.md` states the project rule: *"If we hold the drivers for a piece of period hardware, we plan a personality for it."* That is necessary but not sufficient — on its own it says nothing about whether a personality can be *tested*, whether it collides with another, or whether the fitted ROM still supports it.

The five tests

1. **Driver held?** — the existing rule. A driver means no Osborne-side software work and a known-good test suite.
2. **Interface reachable?** — Z80 socket, char-gen socket, or a port (P1/P2/P3/P5), without cutting traces.
3. **Testable?** — is there software we can *run* against it (MAME or hardware)?
4. **Address-space arbitration** — does it collide with another personality?
5. **ROM-gated?** — does the fitted ROM still support it? (#86 is closed; the gate is built)

Stacking is not the constraint — addresses are

Worth stating plainly, because it is easy to get wrong: **virtual interposers stack.** The maintainer's own machine carries a real SCREEN-PAC *and* a real CoPower-88 at the same time, which is the existence proof — and in the virtual case it is easier still, because every personality is firmware rather than a board. A personality that would have been a Z80-socket interposer in period (the ACT hard disk, the CoPower-88, the TRA video board) is therefore **not** excluded by occupancy: the interposer is already in that socket, so it can simply *be* the device.

The real constraint is **decode**. Two personalities cannot claim the same addresses at once, and the O1′s decode is famously loose — the "4 port boundary" the hard-disk controllers demand is

the same coarse two-line decode the rest of the machine uses. So the arbitration question for each candidate is: *which addresses does the driver touch, and who else wants them?*

The map

Personality	Driver	Interface	Addresses it claims	ROM gate	Tier
`bus-analyzer`	—	Z80 socket	none (passive)	—	**0 — prerequisite for all**
`rt60a-rtc`	ABI documented (#7)	mailbox; **no IEEE handshake**	the `0x0040` block is *software's*, not ours	fine	**1**
`drive-c-gpib`	`DCL`/`DFD.SPR` ✓	IEEE-488 (P3), PIA `2900`–`2903`	GPIB **device address**	**not under OZROM**	**1**
`virtual-printer`	ROM IEEE driver ✓	IEEE-488 (P3)	**device 3**	Centronics survives OZROM	**1**
`floppy-emulator`	—	floppy bus	none (interrupts the Shugart bus)	—	**2**
`screenpac-video`	—	Z80 + UA15 tap	video window + `2400H` bit	—	**2**
`zimodem-fork`	—	P1 (TTL)	ACIA `2A00`/`2A01`	—	**2**
`acb4000-scsi`	`HARD`/`PORTCHNG` ✓	*virtual* Z80 interposer (port block)	a relocatable 4-port window (default `30h`, #8)	?	**3**
`copower88-shim`	drivers + ROM ✓	*virtual* Z80 interposer	`0x7E`/`0x7F`	?	**3 — last**
`kvm-console`	composition	video + keyboard	none	—	**3**

The open question that decides the IEEE row

Three personalities want the IEEE-488 port: Drive C, the virtual printer and the RT-60A. **GPIB is an addressed bus** — the printer is *device 3* (OZROM's appendix: *"IEEE-488 as IOBYTE device 3 now null device"*) — so if Drive C and the RT-60A answer different addresses, all three can coexist, and

the interposer can multiplex because it already sees the PIA at `2900–2903`. **That is a deliverable of #6, and it is the difference between "one personality at a time" and "coexist".**

What this section is not

It is not a decision record, and nothing above has been chosen. The tier ordering is a proposal for sequencing. **One row is now decided:** the storage question (#8) resolved to the `acb4000-scsi` personality — emulate the Adaptec ACB-4000, not the ACT (`docs/harddisk-decision.md`). Drive C remains the easiest to build **standalone** (#84), which is a different claim from "the only one worth emulating".

Prior art: Z80 bus interposers

Status: Partially solved. The survey is complete and every entry carries its licence, and §1.6 retires the timing claim that carried the most risk; what is left is a conflict the survey records rather than resolves (§3′s level-shifting advice against picoZ80′s shipping board), one premise §7 still has to check against the wait generator's own decode, and one decision for the maintainer (§8′s licence-posture question).

Input to the interposer (`01NEXT Upgrade`, Rev A). Prior art for an `RP2354B` interposer between a 4 MHz, 5 V Z80 and its socket, with passive capture + active drive + USB/WiFi streaming. All claims cited; sources verified 2026-10-08.

Serves #11 (the interposer), #12 (the electrical/mechanical interface spec), #17 (the bus analyzer, the closest personality to a survey of sniffers) and #259 (the one-board collapse, which re-opened the prior art).

Two entries carry more weight than the rest, and each answers a different half of the interposer's problem.

- **§1.6, picoZ80 / pico6502, is the closest prior art to the interposer's central timing claim.** It is a *drop-in Z80/6502 CPU replacement* — not an interposer — with a **cycle-accurate PIO bus interface clocked by the target**, 30 releases and four machine families behind it. It makes the interposer's timing claim an existence proof rather than a hope, and it is the most mature board in this survey.
- **§1.7, z80_pico, is the only project here that answers the question the interposer actually has:** what to do when you must *follow* somebody else's clock. Its answer is that the /WAIT assertion has to be **combinational, not programmed**, and it is the only entry that measures what stretching every cycle costs (three wait states halve the machine).

Three consequences run through the document: §3′s level-shifting advice now has two counter-examples and three competing answers, §2.1′s "directly copyable" claim does not survive §8′s licence check, and §7 gains both a clocking idea and a wait-state strategy. **§8 is the one to read before reusing anything here** — two of the survey's reuse claims are GPL-3.0 / CC-BY-NC-SA and are not copyable.

---

1. RP2040/RP2350-based bus capture / CPU-emulation projects

1.1 gusmanb/logicanalyzer (24-ch, 100 Msps Pico logic analyzer)

- URL: https://github.com/gusmanb/logicanalyzer
- MCU: RP2040 (Pico / Pico W; Pico W supports WiFi streaming).
- Approach: PIO + DMA capture, up to 24 channels into RP2040 RAM; pre/post trigger; edge or pattern trigger; 100 Msps, 32K samples. Separate fast level-shifter board required for 5 V targets.
- Level shifting: dedicated shifter PCB (KiCad in repo). Documented **gusmanb/logicanalyzer Rev-A** erratum: J1/J2 footprints exchanged (inputs<->outputs); recovered by flipping the symmetric board. *(Someone else's board: not the interposer's Rev A.)*
- Capture vs drive: capture only.
- Key lessons:
  - Trigger logic must live in PIO; naive GPIO-compare loops cost ~16 cycles/sample.
  - 100 Msps is burst-to-RAM via DMA, NOT continuous USB streaming; capture depth is the limit.
- Relevance: proves PIO samples >20 lines far above 4 MHz Z80 speeds; its Pico W WiFi mode is our streaming precedent.

1.2 AESilky/sd-dkr

- URL: https://github.com/AESilky/sd-dkr
- MCU: RP2040. Disk, keyboard, RTC as a device on a Z80-class parallel bus (device side, not interposer).
- Relevance: RP2040 meeting Z80 bus-peripheral timing.

1.3 lambdamikel/picoram-ultimate & picoram6116

- URLs: https://github.com/lambdamikel/picoram-ultimate, https://github.com/lambdamikel/picoram6116
- MCU: RP2040. SRAM emulator + SD interface for vintage SBCs (incl. 6116 SRAM emulation for the Z80-based Microprofessor MPF-1). Pico sits in the memory socket and drives the data bus synchronously with Z80 cycles.
- Key lesson: bare RP2040 meets ~4 MHz Z80 memory-cycle timing without glue logic; evidence RP2350 meets a Z80 T-state budget.

1.4 74HC138/RP-Floppy

- URL: https://github.com/74HC138/RP-Floppy
- MCU: RP2040. Bitbanged PIO floppy controller (early stage, GPL-3).
- Relevance: PIO MFM/flux handling reference for the floppy personality.

1.5 Flee-Time/FloppyController

- URL: https://github.com/Flee-Time/FloppyController
- MCU: RP2040 floppy controller speaking the Greaseweazle protocol, plus USB MSC. Second datapoint for PIO flux capture.

1.6 picoZ80 / pico6502 (Philip Smart, engineers@work) — the field's most mature example

- URL: https://git.eaw.app/eaw/pico (Gitea) · https://eaw.app/picoz80/
- MCU: **RP2350B** (bare part, not a module) + **8 MB PSRAM** + 16 MB flash, with an **ESP32-S3-PICO-1** co-processor for WiFi/BT/SD-card/web UI.
- Approach: **a drop-in CPU replacement, not an interposer.** The board plugs into the Z80 (or 6502) DIP-40 socket and *is* the CPU: three PIO blocks take cycle-accurate control of the address, data and control buses, and the RP2350 emulates the instruction stream behind them. 54x19 mm board, parts on both sides, DIP-40 extender footprint; v2.5 fab files are in-tree.
- Maturity: **30 releases** (latest v3.434, 2026-08-09), and the machine list is not a demo — Sharp MZ-700/800/80A/80B/1500/2000/2200/2500, Tatung Einstein TC-01, Amstrad PCW-9512, plus a generic `BaseZ80` and an **RC2014** persona.
- **The PIO technique** (`projects/tzpuPico/src/z80.pio`, ~30 KB):
  - **One PIO program per Z80 bus-cycle type** — normal fetch, interrupt acknowledge, NMI acknowledge, and the memory/I/O read and write cycles — with the **T-states written out as comments** (`T1`…`T7`): the published Z80 bus timing diagram transcribed state by state.
  - **Every T-state is one Z80 clock.** The program advances with `wait 1 gpio <CLK_S>` / `wait 0 gpio <CLK_S>` against a **derived Z80 clock on a GPIO** (GP35), rather than dividing the 150 MHz system clock. This is the architectural idea worth understanding: *the PIO is clocked by the target*, so T-state accuracy does not depend on the PIO's own clock budget.
  - **Wait states are real**: `jmp pin` on the `WAIT` line inserts TW and loops, so the host sees the Z80 stretch a cycle exactly as it should.
  - **Inter-SM handshakes through PIO IRQs** (`irq set 1` / `wait 0 irq 1`) to sequence address, data and control state machines.
  - Pin map: address GP0-15, data GP16-23, control GP24-33 (RD/WR/BUSACK/HALT/M1/RFSH/MREQ/IORQ/BUSRQ/INT), derived clock GP35.
- **The C side** (`src/Z80CPU.c`): the instruction core runs in C and **polls the PIO** — `z80_run` reads `pio2_hw->irq` for a /RESET from PIO SM3 — so the PIO owns the bus and the CPU code blocks on it. Its `PICOMOD.md` records Cortex-M33 micro-optimisations with measured costs (indirect callback ~5 cycles vs direct ~2; call overhead ~10-15 cycles; a local cycle counter in the run loop ~3 cycles/instruction), useful to anyone writing a hot loop on this core.
- **Virtual device framework**: 57 C files of period-peripheral models — `PPI8255`, `PIT8253`, `Z80SIO`, `uPD765`, `WD1770`/`WD1773`, cassette, quick disk, **SASI** (the SCSI predecessor) — plus per-machine drivers. Configuration is a **`config.json` on the SD card** (245 of them in-tree): memory map, ROM images and driver selection are runtime data, not compile-time choices. That is a direct alternative to our compile-time personalities and to the mailbox protocol for configuration.
- Electrical: ~45 series resistors in two symmetric banks (R1-R19, R28-R45) and **seven ICs total** on the placement file — i.e. **no bus-transceiver array**. See §3, where this contradicts our own advice.
- **Licence: firmware GPL v3, hardware and docs CC BY-NC-SA 4.0, plus NOTICE terms on attribution, trademarks and passing-off. Not reusable — §8.**

- Key lessons:
  - **A cycle-accurate PIO Z80 bus on RP2350 is not a hope; it ships.** That retires the largest open risk in this project's design (§7).
  - **Clock the PIO from the target's clock** wherever you can see it. A CPU replacement owns the clock; an interposer cannot — but if the interposer can see Z80 CLK, a PIO clocked from it makes T-states exact rather than asserted.
  - **One PIO program per cycle type, with the T-states in comments**, is a reviewable decomposition and the natural way to check work against a timing diagram.
  - A per-machine driver framework plus a JSON memory map is the shape our `firmware/` personalities are groping toward.

1.7 z80_pico (tendai22) — small, unfinished, and the only worked answer to our specific problem

- URL: https://github.com/tendai22/z80_pico
- MCU: **RP2040** (Pico). 5 commits, 13 stars, one target machine (SBC8080), no releases, and **no licence file at all** — see §8.
- Approach: the **third** architecture in this document. Not a replacement and not an interposer — a *host that subjugates the CPU* (the RetroShield shape, §2.3): the Pico generates the Z80 clock (GPOUT0 on GP21), holds the whole machine's 64 KB of ROM/RAM in RP2040 SRAM as a C array, emulates an **i8251** on UART0, and **stretches every memory cycle with /WAIT**. The author's own verdict: *"far from elegant design, but I dare to do it for my personal intention."*
- **What it is worth reading for, and it is the only project here that answers it:**
  - **Software cannot assert /WAIT in time.** *"Even if RP2040 CPU is fast, it is still hard to issue /WAIT in time to make Z80 surely to start wait state. A half of 74HC74, D-FF takes falling edge of /MREQ to flip its output /Q, directly connecting to /WAIT input of Z80 CPU."* The assertion is **combinational, not programmed** — and `/RFSH` has to gate the D-FF's clock input, or the refresh cycle's /MREQ falling edge is mistaken for a memory cycle. **Every other project in this document dodges this by *generating* the Z80 clock** (§1.6 included). The interposer is the only architecture that must *follow* one, so this is the only worked answer to our version of the problem.
  - **Releasing /WAIT and the data bus is a hardware problem too.** *"I am glad to adopt a hardware logic (DO buffer and its /OE signal connected to /RD) to handle this issue. No software provision is needed. If we implement it as software, this issue will become much more confusing."* That is the same hazard `docs/picoem-bridge.md` §4 met from the other side: *"the idle state has to be driven, not left alone."*
  - **The cost of stretching every cycle, measured.** Z80 clock 3.8 MHz, *"effectively about 2 MHz, with 3 WAIT cycles"*; `ASCIIART.BAS` in MSBASIC took **8 min 23 s**. Three wait states per memory cycle **halves the machine**. This is the number that says the interposer must be **transparent when it is passive** — a design that stretches every cycle is not an interposer, it is a new machine wearing one.
- **Level shifting**, and it refines §3 rather than repeating it: eight level shifters on the 5 V⟶3.3 V **input** path and **none** on the 3.3 V⟶5 V output path — *"3.3 V output can drive 5 V CMOS gate*

inputs, and we do not have any critical issues, so it is okay." Shift what you receive; be deliberate about what you drive.

- **Pin budget**, for comparison with our 595/165 shift fabric and picoZ80′s direct mapping (§1.6): 8 bits in + 8 bits out (GPIO0-7 DI, GPIO8-15 DO), with **six buffers/latches** — address high, address low, data in, status, controls, data out — selected onto the input port by a **74HC138 3-to-8 decoder** on GP18-20. The author notes a decoder "can decrease the necessary number of GPIOs"; the real cost is that every bus read becomes a multiplexed *sequence* rather than a parallel sample.
- Also present: **INTA support** (`M1 && IORQ` → stretch, then put `0xFF`/RST 7 on the data bus), and a `rom2c` ROM-to-C-array tool — the CPU-in-a-jar pattern, which is not ours.
- Key lesson: **the interesting part of this project is its limitations.** It is the smallest and least finished thing in this document, and it is the only one that says out loud what it costs to get bus timing wrong by a factor of two.

2.1 AtomBusMon (David Banks / hoglet67)

- URL: https://github.com/hoglet67/AtomBusMon (stardot.org.uk)
- Approach: "ICE" adapters piggybacking the 40-pin CPU socket (6502, Z80, 6809, 8080…); Z80 adapter uses 6x 74LVC4245 level shifters and turned-pin (Preci-Dip) headers; single-step, breakpoints, full bus trace.
- Errata accumulated across spins ("R10,R12,R13 should not be fitted", "C15 not fitted"); extra socket needed for stacking height.
- Key lesson: THE canonical socket-piggyback ICE design; its Z80 adapter BOM and shifter choice are directly copyable.

2.2 BusRaider (Peter Tandler / Rob Dobson r12a)

- URL: https://github.com/robdobsn/BusRaider
- Approach: Raspberry Pi Zero on the RC2014 (Z80) bus; can snoop and DRIVE the bus; `rawBusClockDisable` / `rawBusTake` model — stop the Z80 clock, then walk the bus at leisure.
- Key lessons:
  - Clock control converts impossible real-time problems into trivial ones. Add WAIT-stretch / clock-gate hooks to Rev A.
  - A debugging session found swapped PAGE_WR/PGEN_WR lines; a comparative trace against a known-good board would have caught it earlier — build golden-trace diff tooling, not just tracing.

2.3 RetroShield (8bitforce) and skx/z80retroshield

- URLs: https://github.com/8bit-1/retroshield (Arduino), https://github.com/skx/z80retroshield
- Approach: Arduino/Teensy drives a real Z80′s clock and buses, emulating memory/IO in software (CPU-in-a-jar). skx rewrote the upstream sample into a standalone library after finding mixed concerns (buttons/LCD/SPI-RAM tangled with bus driving).
- Key lesson: keep the bus-driver firmware layer a clean standalone module; also proves ~MHz Z80s can be driven by MCUs.

3. Electrical design lessons

- Level shifting observed in the wild: 74LVC4245 x6 (AtomBusMon), dedicated fast-shifter boards (logicanalyzer), 74LVC245-class parts recommended for inputs; 74HCT245 or dual-supply parts for outputs.
- RP2040/RP2350 GPIOs are NOT 5 V tolerant; do not rely on series-resistor hacks for a product-quality board. **Counter-example, and it is the most mature board in the field** (§1.6): picoZ80 appears to carry ~45 series resistors and **no** level-shifting or transceiver ICs, on a board that sits in a 5 V Z80 socket and has shipped 30 releases — its placement file lists seven ICs (U2-U7) and no 74-series array. This bullet and that board cannot both be right. **Resolve it from their schematic and from the RP2350 datasheet's GPIO absolute-maximum ratings, not from either document's advice — and note that the placement file has no values, so "series resistors" is an inference from 45 symmetric designators, not a reading.**
- **A third answer, and it splits the problem in two** (§1.7): z80_pico shifts the 5 V⟶3.3 V **inputs** (8 level shifters) and shifts **nothing** on the 3.3 V⟶5 V outputs, on the grounds that *"3.3 V output can drive 5 V CMOS gate inputs, and we do not have any critical issues."* So the field's answers are: series resistors everywhere (picoZ80), transceivers everywhere (AtomBusMon), or **shift what you receive and be deliberate about what you drive**. Note the caveat that makes the third one work and is easy to miss: a 74HC input at VCC 5 V has VIH ≈ 3.15 V, so a 3.3 V drive is *just* inside spec — which is not the same as having margin, and is worth measuring on the O1 rather than assuming.
- Verify whether the Osborne 1's Z80 is NMOS (VIH 2.0 V -> 3.3 V drive OK) or CMOS Z84C00 (VIH ~3.5 V -> needs HCT/dual-supply) before committing the output stage. [Check against machine photos in research/pictures.]
- Power: don't feed WiFi transmit peaks from the CPU socket 5 V pin without checking regulator headroom; prefer USB-powered Pico with common ground + generous local decoupling (AtomBusMon: 12x 100 nF
  - bulk).
- Mechanical: turned-pin headers; expect stacking-height problems inside the O1 case; keep profile minimal and test-fit early.

4. PIO sampling & streaming

- Clock-synchronous sampling (sample on the clock edge qualified by MREQ/IORQ) yields compact per-cycle records; oversampling is affordable (4 MHz bus, >=25x PIO headroom per logicanalyzer's 100 Msps / 24 ch).
- USB FS budget ~1 MB/s is insufficient for raw every-cycle capture at 4 MHz. Mitigations, in order:

  a) PIO-side filtering (M1-only, address windows, triggers) — trigger logic must live in PIO;
  b) RAM burst capture with pre/post trigger;
  c) WiFi streaming as a peer path.
- Double-buffered DMA from day one.

5. Floppy & GPIB references

- FlashFloppy/Gotek (Keir Fraser) is STM32, NOT Pico — do not plan around porting it; instead track 74HC138/RP-Floppy and Flee-Time/FloppyController for PIO MFM, and Keir's Disk-Utilities (disk-analyse) on the host side.
- NODISKEMU (https://github.com/nils-eilers/NODISKEMU) and cbmSD (http://cbmsteve.ca/cbmsd/) are AVR-based; their IEEE-488 state machines are the reference to port to PIO for the Drive C personality.

6. Documented errata / "what I'd do differently"

- logicanalyzer: J1/J2 footprints swapped in first fab run.
- AtomBusMon Z80 adapter: BOM fit/no-fit errata across spins; stacking-height fix.
- skx/z80retroshield: rewrote upstream into a standalone library — separation of concerns matters.
- BusRaider: golden-trace diffing would have caught swapped lines.

7. Synthesis: recommendations for Rev A

1. Architecture: RP2354B BETWEEN Z80 and socket (AtomBusMon-style 40-pin DIP piggyback), real Z80 in a top socket so passive mode is electrically transparent. Two PIO banks: capture (clock-edge-sync push of addr/data/status) and drive (data bus + WAIT/NMI/INT/BUSRQ).
2. Level shifting: 74LVC245 inputs + 74LVC8T245/74HCT245 outputs, following AtomBusMon precedent.
3. Capture rate is a non-problem; spend the effort on PIO triggers.
4. Streaming: PIO filtering -> RAM burst -> USB/WiFi drain.
5. Active drive: WAIT-stretch and clock-gating hooks (BusRaider model) — being able to stop the clock is a superpower. 5b. **The target clock is already in the design, so the prior art settles a question Rev B had already answered (§1.6).** `hardware/interposer-upgrade/revb/README.md` allocates **GP21 = CLK, "direct — PIO timing reference"** — and the machine's clock is a *mainboard* signal, 15.9744 MHz ÷ 4 through the UD3 LS161 counter (`o1-memory-io-map.md` §1), not the Z80′s own oscillator. So the interposer is loading a distributed logic output rather than a crystal, which is the easy case. picoZ80 proves the technique; the open item is therefore not *whether* to use the clock but **what the state machine does with it**: a PIO *clocked* by CLK executes one instruction per T-state (picoZ80′s constraint, and why its programs are one `set pins` per T-state), while a PIO on the system clock *observing* CLK has ~37 instructions per T-state but must track cycle phase in firmware. The Rev B README does not say which, and the two are different programs. 5c. **And /WAIT should be scheduled rather than raced — a structural argument, not a cautious one.** Rev B asserts /WAIT *after* observing `ADDR_HIT` (GP23), so it is a **reaction**: it must fit inside the window on every cycle, worst case, across temperature, voltage and part tolerance. An assertion derived from the clock phase can be made in **T1**, before the address has even settled, because the interposer knows from the clock that this cycle is one it owns. A race has a statistical margin; a schedule has a guaranteed one.

The window looks comfortable and the claim is not obviously wrong, but the margin is **inferred, not measured**: `o1-memory-io-map.md` §1 attaches the machine's wait states to **RAM accesses** (≈375 ns for non-M1, 188 ns for the first M1) and says ROM executes with no added delay — and it says nothing at all about **I/O cycles**, which is what the interposer's `30h–33h` window actually is. *If* the I/O cycle is also unstretched, `ADDR_HIT` early in T1 leaves ~1.5 T-states (≈375 ns at 4 MHz) to the falling edge of T2 against the README's "~50 ns", a 7× margin. **That premise needs checking against the wait generator's own decode on sheet 2/9, not assumed** — it is the number the margin rests on.

And **50 ns is one term in an unmeasured sum** — it is the PIO's response, and the path also contains the 688′s propagation, whatever converts its 5 V output down to GP23′s 3.3 V, the PIO's two-cycle input synchroniser, and the OC buffer's turn-on. That sum is the one number the live-response path rests on, and it has not been measured. 5d. **The machine already stretches its own cycles, so /WAIT is a wire-OR — with three consequences.** `o1-memory-io-map.md` §1: RAM accesses insert ≈375 ns of wait for non-M1 cycles and 188 ns for the first M1, while ROM executes with no added delay. So (a) the interposer's /WAIT drive **must** be open-collector or otherwise OR-able, and Rev B's "fast OC buffer" on GP22 is the right call;

(b) the interposer's T-state machine must tolerate TW states **it did not insert**, because it cannot assume a cycle is three T-states long; and (c) **MAME omits the machine's wait states** (`o1-mainboard-schematic.md` §8), so this interaction cannot be exercised in emulation at all — which makes it a hardware-gated gap in its own right, and the one behaviour a race is least able to survive.

6. Power: USB-powered Pico, common ground, heavy decoupling.
7. Mechanical: turned-pin headers, minimal profile, early test-fit in the O1 case.
8. Process: publish BOM with fit/no-fit notes; expect >=1 footprint erratum; build trace-compare tooling early; keep the bus-driver firmware layer standalone.

8. Licences, and what "prior art" actually lets us reuse

§2.1 says of AtomBusMon that *"its Z80 adapter BOM and shifter choice are directly copyable"*. That is a claim about **engineering**; **licensing** is a separate question, and it has been checked for every entry that could plausibly be reused. All three need qualifying:

Project	Licence	What it permits
picoZ80 / pico6502 (§1.6)	firmware **GPL v3**; hardware and docs **CC BY-NC-SA 4.0**; a NOTICE adding attribution, trademark and anti-passing-off terms	**Study it; do not copy it.** GPL v3 is one-way: code taken into o1next would relicense our firmware as GPL v3. CC BY-NC-SA forbids commercial use *and* forces ShareAlike on derivatives, which is incompatible with this repository's **MIT** licence, with a Phase 5 reproduction (#49) and with the deliverable tree (#133–#140)
AtomBusMon (§2.1)	**GPL-3.0**	The same direction of problem, one step milder: a *schematic* is a copyrightable work, so "the BOM and shifter choice are directly copyable" is true of the **part numbers** (facts) and false of the **design files** (expression). The turned-pin piggyback idea and the wiki are fair to learn from
z80_pico (§1.7)	**none — the repository has no licence file**	**The strictest case, and the one people get wrong.** *"It is on GitHub" is not a licence.* With no licence all rights are reserved; GitHub's terms grant viewing and forking through GitHub's own features, not reuse in our project. The practical answer is the same as picoZ80 — learn from it, copy nothing — arrived at for an entirely different reason

What that leaves, and it is not nothing:

- **Ideas, methods and facts are not copyrightable.** The technique in §1.6 — clock a PIO from the target's clock, one program per cycle type, `jmp pin` on /WAIT, IRQ handshakes between

state machines — is fair to learn and to reimplement in our own code. What we may not do is copy their `.pio` file, their C, their KiCad or their documentation.
- **Existence proofs are facts.** "A cycle-accurate PIO Z80 bus on RP2350 ships, with 30 releases" is usable in an argument and needs no licence at all.
- **Asking is a real option, and it is invited.** picoZ80's NOTICE ends with contact details for exactly this, including commercial licensing. If any part of it turns out to be worth more than the idea, the route is a message — and this repository should record the answer either way.
- **Put the licence in the entry, and check it before writing "copyable".** A part number is a fact and travels; a design file is expression and does not.

Open question for the maintainer

picoZ80's licence posture (non-commercial hardware, GPL firmware, trademark and anti-passing-off terms) is the opposite of this project's MIT default, and it is worth a deliberate decision rather than a default: **do we want o1next to remain MIT, or is a copyleft/non-commercial posture acceptable for parts of it?** The answer changes what can be borrowed, and §1.6 is the first entry where the question actually bites.

Open questions

Status: Solved — as a status review: every open research question is tracked here with its evidence and what is left, cross-checked against the `research`-labelled issues and against each document's own status line. Last reconciled with the documents at the `docs/` readability sweep (#200).

This is the single place that says, for every open research question, **whether it is answered, by what evidence, and what is left**. Where this file and a document disagree, the document is the authority — it holds the measurement — and this file is the one that is wrong.

Phase 0 reads **22 of its 23 sub-issues closed**, counted from the issue tree (§F); **#39** is the last one open.

Status words (same vocabulary as `docs/README.md`):

Status	Means
Solved	Answered, and the answer is checkable from a doc
Partially solved	The main shape is known; a *named* sub-question is not
Open	Not answered
Superseded	Stopped mattering — a better route was found, or a decision was taken

A gap is only **Solved** if the evidence is written down. "We know this" is not a status.

A **gap** and the **document** that carries it answer different questions, so their words can differ without disagreeing: a gap is *Solved* when the question is answered even where the doc is *Partially*

solved because a second, smaller sub-question is still open (gap 8′s RT-60A is the worked example — the register map is answered, the doc is not). Where the two differ, the **What's left** column names the missing thing, and that is what reconciles them.

Summary

	Count
Solved	9 of the original 15
Partially solved	3
Open	1
Superseded	2

What actually blocks the next phase — everything else can wait:

1. **#39** — the identity of the three target machines (user action; blocks gap 12).
2. ~~**Drive C's command set**~~ — **closed 2026-10-11 (#6).** The transport and the device contract are measured from the disassembly *and* a trace of the real software, and a model of the box makes the 1983 loader complete and PIP write to `C:`. The DPB came out of `DCL.COM` (#145), and `DFD.SPR` turned out to be a replacement CCP rather than the transport, so the disassembly it was waiting on is off every critical path. What is left is **#84′s**, not research's: how the box resolves banks 0–2 (hardware-gated) and whether Drive C, the printer at device 3 and the RT-60A can share one bus. The build spec is `docs/drive-c-design.md`.
3. **The hard-disk personality (#179)** — the controller *decision* is closed: emulate the **Adaptec ACB-4000** inside the Media Distributing MD-10, not the ACT (#8), and **#178 is closed with it** — the drive table's layout and the DPB's values are both read (`harddisk-decision.md` §7.2–§7.3, `md10-system.md` §9). The personality is built and host-tested, and the MD-10 system it needs boots to a prompt with three partitions (`md10-system.md` §1, §7). What is left is **who copies the static DPB and descriptor tables into the running system** (`harddisk-decision.md` §10 item 1, `md10-system.md` §9.1) and pointing the MAME harness at this personality rather than at MAME's generic SCSI model.
4. ~~**The RT-60A ABI (#7)** — decide the contract, then implement it in #18.~~ **Done 2026-10-11.** The ABI is written as a contract in `rt60a-analysis.md` (pointer block, sentinel/invalidation rules) and #18 implements it. The hardware handshake is retired as a known unknown.
5. ~~**Does the DD upgrade read foreign formats? (#95)**~~ **Solved and closed 2026-10-11.** The answer is in the OCC CBIOS source — a DPB + sector-skew table (`occbio95.asm`) whose capacities match the manual's list exactly, selected by a monitor-ROM density probe — and Osborne SD is confirmed end-to-end in MAME. The four foreign formats are **taken as written, not synthesised**: they are only needed to read original foreign media *on the O1*, and foreign media is converted on the modern side instead. `dd-upgrade-formats.md`.
6. ~~**OZROM 1E as tooling (#81)**~~ **Solved and closed**: `ozrom-1e.md` indexes the manual, measures it on all three machines, and records the column switch's mechanism from the ROM. Of its four residues one is live — **#167**, driving the column keys from the harness — and **#168** (a complete

Utilities disk) is closed: we already had one, and the "290 of 400 sectors erased" that justified the hunt was a `cpmfs.py` sector-skew bug (#170).

Everything else is either done, or is research that can proceed in parallel. **#79 is closed and no longer blocking**: the decision taken is not to chase the Nuevo path, and to build the video personality from the ScreenPac design plus OZROM's soft 52/80/104-column switch.

A. The original gaps

#	Gap	Status	Evidence	Issues	What's left
1	Exact memory & I/O map, free decode windows	**Solved**	`o1-memory-io-map.md` §2–§7 (schematic), confirmed from the ROM source in `o1-rom-source.md`, measured on the running machine by `tools/check_o1_map.sh`	#40, #62, #64, #72	Decode *granularity* is still schematic-level — the chip selects check A11 plus one of A8/A9/A10 (`o1-mainboard-schematic.md` §7, §7a). Of the ranges this row used to call free, only three page-aligned blocks are selected by *nothing*: `2000–20FF`, `2400–24FF` and `2800–28FF`; `2800–28FF` is the recommended virtual-peripheral base (`o1-memory-io-map.md` §7)
2	NMOS vs CMOS Z80 across the target machines	**Superseded**	Resolved by history: the CMOS Z84C00 shipped in 1985, O1 production ended in 1983 — every O1 is NMOS. Interposer	#39	#39's remainder: machine IDs and date codes, for the record
[illegible]	[illegible]	**Solved**	[illegible]	#179, #191	[illegible]

B. Questions that appeared since the original review

B1. Hard-disk products — two in the archive, and seven routes advertised in total

The original gap 9 assumed a single OCC1 hard disk. The archive drop (#71) turned up a second, unrelated one. Confirmed from their own strings:

	ACT hard disk	Media Distributing / Adaptec
Source	`research/harddisk/OCC1_HARDDISK.IMD`	`research/adaptec/*.img`
Vendor string	`AUSTRALIAN COMPUTER & TELECOMMUNICATIONS COPYRIGHT (C) 1982`	`Copyright (c) September 1983, Media Distributing`
Versions	`Version 31.05.82`, `Version 9-02-81`	`HARD version 1.93`, `PORTCHNG version 1.92`, `PREP version 1.93`
Controller	its own; `Enter disk controller address (`	Adaptec ACB-4000; `Enter starting port address in hex, must be on a 4 port boundary`
Utilities	`BOOTHD`, `LOADBIOS`, `HARDBIOS.HEX`, `MOVCPM10/5/F`, `DISKEDIT`, `DISKTEST`, `RESTORE`, `SAVEFILE`	`MD10.COM`, `MD20.COM`, `HARD.COM`, `HARDBIOS.SPR`, `HARDBDOS.SPR`, `HARDCCP.SPR`, `HD00/HD04BIOS.SPR`, `PREP.COM`, `PORTCHNG.COM`, `DRIVETBL.DAT`

So the three routes **represented in the archive** are the **ACT hard disk**, the **Media Distributing/Adaptec hard disk**, and **Drive C** (IEEE-488). Both hard disks use a *port-relocatable* controller — which matters, because the O1′s I/O decode is sloppy, so "4 port boundary" is the same two-address-line decode the rest of the machine uses.

Four more routes appear only in the magazine archive, with no software or media of ours behind them — recorded because a survey of "what storage existed" is incomplete without them, and because none is buildable from what we hold:

- **Corvus hard disks**, reached through a link program — `CLINK2`, built from `CLINK2K2.ASM` and assembled by the owner with `ASM.COM` and `LOAD.COM`. *The Portable Companion* #11 (October 1983), PDF p. 54 = printed p. 52, "**Doubling hard disk memory**" by Bob Gras (reprinted from the OSUG newsletter, Anchorage), is a patch for exactly that program, needed because DD leaves less memory for it (`docs/dd-upgrade-formats.md`). No Corvus software or media held.
- **Osmosis OSMOS 7**, advertised as "Winchester disks … 5¼, 10½, 15¾ & 21 MByte units - a 42 Megabyte max capacity", $2475–$3650 — on the same page as the OSMOS 5 80-column board,

which itself "**includes a Hard Disk interface for future plug-in expansion**" (`docs/o1-upgrade-boards.md`). No media held.

- **Trantor TSL-5 / TSL-10** — *The Portable Companion* #5 (February/March 1983), PDF p. 28: "unpack your disk, **plug it into the IEEE port**, run a simple 5 minute installation procedure", 5 MB $2995 / 10 MB $3395, supplied with BIOS, back-up and retrieve utilities and cable (`research/technical-articles/TPC-Issue05-1983-pp28-trantor-tsl-hard-disk-ad.pdf`). This is the line `docs/drive-c-protocol.md` records the Drive C vendor as reselling at a discount; the advertisement is what pins its interface to the **IEEE bus**. No media held.
- **Helix** (Micro Computers of New Orleans) — *The Portable Companion* #5, PDF p. 98, "**Winchesters Now Available For The Osborne 1**", 5/10/15 MB at $2500/$3000/$3500, a "**Z-80 adaptor** card [that] leaves external ports free" and a ribbon cable that "disconnects for portability" (`research/technical-articles/TPC-Issue05-1983-pp98-tra-80-100-column-video-and-winchester-ads.pdf`). No media held.

One of the two archived products now has a name. The Mega-byte Inc. advertisement in *The Portable Companion* #5 (PDF p. 25) sells the **MD-10** — 10 MB formatted for $2295, "20 and 44 megabyte Winchesters also available" — and its footnote reads *"MD-10 is a registered trademark of **Media Distributing**."* That is the same Media Distributing whose September-1983 disk sits in `research/adaptec/`, and that disk ships `MD10.COM`/`MD20.COM`, matching the 10 and 20 MB models. So the archive's "Media Distributing / Adaptec" entry **is the MD-10**, with an Adaptec ACB-4000 controller inside (`docs/occ1-harddisk-analysis.md`). The ad also fixes the *architecture*: the MD-10 "does not tie up existing OSBORNE ports" and is "significantly faster than other winchester subsystems **which use the IEEE port**" — i.e. it is **not** an IEEE-488 device, unlike Trantor, Drive C and the RT-60A. Taken with the ACT unit's Z80-socket interposer and Helix's "Z-80 adaptor card", that is a pattern worth recording for #84: **several vendors reached the Osborne 1's storage through the CPU socket or a port of their own, not through IEEE.**

So the honest count is **seven** advertised storage routes beyond the floppies — ACT, Media Distributing/Adaptec (**the MD-10**), Drive C, Corvus, OSMOS 7, Trantor TSL-5/TSL-10 and Helix — of which **two** are buildable from what this project holds: **Drive C** (#84), and the **MD-10**, whose ACB-4000 personality is built and host-tested (`acb4000-personality.md`) and whose vendor-produced system boots on the emulated drive (`md10-system.md`).

B2. Character-generator ROM variants — closed, and the two fonts measured

##44 asked for the early-font and Nuevo char-gen variants. Both are captured and catalogued (`o1-rom-variants.md`): `5297c109` (early, paired with BIOS ≤ 1.4) and `6c1eab0d` (late, paired with 1.43/1.44 and used by ScreenPac and Nuevo alike).

They really are different typefaces, and that is measured, not assumed. Read glyph by glyph out of the two images (`tools/romfont.py` decodes both): **93 of 128 glyphs differ** — 0 of the 32 graphics characters, 90 of the 95 printable ASCII, and 3 of the rest — so the late font carries the graphics set over unchanged and changes almost everything else. Only space, the apostrophe and the backtick are shared (`o1-rom-variants.md`, *The two fonts really are different*). Which font is fitted is therefore a visible difference, not a subtle one.

B3. The Nuevo system disk does not boot under MAME — #79, closed
`OS1NUEVO.IMD` carries the Nuevo 80-column CBIOS 1.5, loads, prints its banner, and stops. The pristine upstream copy behaves identically, so it is not a bad conversion. The three plain DD system disks *do* boot the Nuevo machine, at 80 columns (`mame-emulation.md` §2, §4).

Decision: not pursued. Nothing in the Nuevo design has turned up a feature worth the validation cost, so the video personality (#24) is built from the **ScreenPac design plus OZROM's soft 52/80/104-column switch**, and the Nuevo board stays prior art (#73). The residual question — MAME fidelity gap, or a physical-card dependency — is recorded in `mame-emulation.md` §4 and only matters if the Nuevo path is ever reopened.

B4. OZROM 1E — a third-party ROM usable as tooling — #81
It boots under MAME (via BIOS substitution) and its manual documents a software 52/80/104-column switch, a chip-level memory tester, a redefinable keyboard, and its own ROM jump table and memory map. It also **removes the IEEE-488 routines entirely** — the vectors at `013F–0156` now deliberately crash the machine, and *"IEEE-488 AS IOBYTE DEVICE 3 NOW NULL DEVICE"* — so **OZROM and Drive C are mutually exclusive** without reimplementing the driver. It also shrinks the video window to 3 KB (`F000–FBFF`) to make room for its keyboard features, which constrains any 2.0 ROM (#83).

Its four closing residues are routed in `ozrom-1e.md` §9. One is live — **#167**, driving the column switch from the harness, which the harness cannot post because it types plain characters. **#168** (a complete Utilities disk) is closed: the disk we hold is whole, and the "290 of 400 sectors erased" was our own `cpmfs.py` sector-skew bug (§10.2). The jump-table diff against `rom144.asm` is done (§6). One residue is new and unowned: the Utilities disk's own `CRCKLIST.CRC` lists a two-byte CRC per file, and **no standard CRC-16 variant reproduces them**, so per-file verification against the vendor's manifest waits on identifying the algorithm (`ozrom-1e.md` §10.5).

B5. Other open research items

Item	Issue	State
Nuevo + Osmosis upgrade boards as prior art (two products each; neither matches OCC's implementation)	#73	**Closed.** Both halves of both products are written up in `o1-upgrade-boards.md` — the video side there, the DD side in `dd-upgrade-formats.md` — with the prior-art note in `research/inspired-by/README.md`. Its residue, **#173** (the Nuevo BIOS 1.5 presence check), is closed too: the check is a read of a port the Osborne 1 does not decode. Two live residues are recorded in `o1-upgrade-boards.md` §Open and neither has an issue: **one measurement** on a real Nuevo board — `IN A,(04H)` with the card fitted — and **Osmosis's paid technical manual**, which the archive does not hold
Advent Products' **Osbaud** baud-rate generator (#21)	#21	The four connection points are decoded from the review + schematic (6850 pins 4/12, 74LS32 and 74LS04 pin 7); the internal crystal/divider topology is **not** known — needs the instruction sheet or a photo. `o1-modem-serial.md`, *Advent Products "Osbaud"*
Archive provenance TODOs (2 items)	#74	**Closed**
`cpmtools` misreads Kaypro DSDD images	#35	`misc`. Upstream identified: the primary site is `moria.de` (Michael Haardt, 2.23, **email patch** — no GitHub, no tracker), and the practical PR venue is the unofficial `lipro-cpm4l/cpmtools` fork (3 open PRs, 2.21-based).
[illegible]	#[illegible]	[illegible]
[illegible] and the O1-NEXT machine)	281	[illegible]

C. The `research`-labelled issues, and what closes each

After the 2026-10-11 triage the label means **reverse engineering and documentation** again. Five issues left it: **#35** and **#82** (`misc` — upstream contributions), **#97** (`misc` — site and font), **#84** (a build, now `hardware/drive-c-repro/`) and **#52** (Phase 5). **#83** was never research-labelled and has been removed from this table. **#75** is not research-labelled either — it is `documentation`/`deliverable`, the book — and it stays only because it is where these specs are finally consolidated.

Issue	Title	State	What closes it
#1	Phase 0 umbrella	Open	The sub-issue tree (§F) — **22 of 23 closed**; #39 is the last one open
#6	Drive C ⟶ GPIB protocol spec	~~Largely~~ **Closed**	Delivered as `docs/drive-c-protocol.md` and validated end to end. The residue moved to **#84** — bank resolution for `n = 0,1,2`, and the GPIB device-address question
#7	RT-60A ⟶ a time **ABI**, not the hardware	~~Partially~~ **Closed**	The ABI *is* the deliverable and it is written in `rt60a-analysis.md`; #18 implements it
#8	Choose which hard-disk controller to emulate	**Closed**	The written recommendation, taken: the **Adaptec ACB-4000**, one personality rather than two (`harddisk-decision.md` §1–§2, §9.5). Its build is **#179**
#178	Read the MD-10 DPB and `DRIVETBL.DAT`	**Closed — both halves read**	The **drive-table half**: the file's structure (13-byte header, 126 11-byte name records, sentinel, 113 128-byte description records) and the byte⟶field mapping — the nine geometry bytes are the ACB-4000's own `MODE SELECT` drive-parameter block — are in `harddisk-decision.md` §7.2;
~~#33~~	~~Read the MD-10 DPB and `DRIVETBL.DAT`~~ (superseded by #178)	~~Closed~~	~~The **drive-table half**: the file's structure (13-byte header, 126 11-byte name records, sentinel, 113 128-byte description records) and the byte⟶field mapping — the nine geometry bytes are the ACB-4000's own `MODE SELECT` drive-parameter block — are in `harddisk-decision.md` §7.2;~~

Closed, but still carrying the label — the work that got Phase 0 to where it is: **#2, #3, #4, #5, #6, #7, #8, #9, #10, #30, #44, #64, #70, #71, #72, #73, #74, ##79, #81, #95, #165, #168, #173, #178, #198, #245**. Most are attached to #1 as sub-issues, so the parent's progress bar counts them.

D. Triage decisions taken 2026-10-11

The 2026-10-11 triage produced *decisions* rather than close-outs — #3, #4, #5, ##44, #64 and #72 were all closed on it — and the decisions are recorded here because they were reached by discussion and are not re-derivable from a source:

- **#30 folded into #95.** Same question one product apart: the DD half is already answered in `dd-upgrade-formats.md` from the DD upgrade manual, and the G2 format table moves to that doc.
- **#35 and #82 stay open, as `misc`.** They are upstream contributions, not reverse engineering — and #35 is a genuine courtesy fix now that `cpmfs.py` gives us a known-good oracle to diff against.
- **#7 re-scoped to an ABI.** Implement the *documented* memory contract, not the *undocumented* IEEE handshake (see §A gap 8).
- **#8 re-scoped to a decision.** Which controller to emulate — one personality, not two.
- **#97 keeps the machine's own repertoire** — the graphics-set Unicode mapping and the recovered logo art. The modern font is split out as **#107** (`Osborne1nx.ttf`).
- **#84 is a build**, filed beside the other standalones (#47, #56) in `hardware/drive-c-repro/`, not under the interposer.
- **#52 belongs to Phase 5** (#49).
- **The book is ungated.** #75 no longer waits on every research issue; it is a rolling build (#108–#115), with Part I and the appendices buildable today.

E. What is left before hardware design starts

Ordered by what blocks what:

1. **#39** — machine identities, board revisions and ROM markings (user action; unblocks gap 12 — the issue lists the three things actually worth reading).
2. ~~**Finish Drive C's command set** from `DCL`/`DFD.SPR`.~~ **Done 2026-10-11 (#6 closed)** — the protocol, the DPB and the capacity ceiling are in `drive-c-protocol.md`, and the **P3 pinout** came out of the Technical Manual already in the archive (`o1-ieee488-port.md` §6). The `DFD.SPR` disassembly came *off* the list rather than onto it: it is a replacement CCP, which a reproduction inherits. The bank-resolution question is hardware-gated and now sits on **#84**.
3. ~~**Decide the hard-disk controller** (#8): characterise both and pick one to emulate.~~ **Closed** — the **Adaptec ACB-4000** inside the Media Distributing MD-10, not the ACT (`harddisk-decision.md`). Its two sub-issues: **#178 is closed** — the drive table's layout is read (§7.2) and the DPB's values turned out to be static data inside `HARD.COM` (§7.3, `md10-system.md` §9) — and **#179 is open**: the personality is built and host-tested, the MD-10 system boots to a prompt

with three partitions, and what is left is who copies the static tables into the running system and pointing the MAME harness at the personality rather than at MAME's generic SCSI model.

4. ~~**Write the RT-60A ABI contract** (#7), then implement it in #18.~~ **Written 2026-10-11** (`rt60a-analysis.md`); #18 implements it.
5. ~~**#73 — the DD halves of the Nuevo and Osmosis boards.**~~ **Closed**: written up in `o1-upgrade-boards.md` and `dd-upgrade-formats.md`, with the Nuevo presence check (**#173**) closed with it — it is a read of a port the Osborne 1 does not decode. Two residues remain and neither has an issue: **one measurement** on a real Nuevo board (`IN A,(04H)` with the card fitted) and **Osmosis's paid technical manual**, which the archive does not hold (`o1-upgrade-boards.md` §Open).

The book (#75) is **no longer gated on all of this**: it is a rolling build whose pieces are tracked as #108–#115, and Part I plus the appendices are buildable now. ##11/#12 (interposer design) have everything they need.

F. The issue tree (added 2026-10-11)

Phase 0 was an umbrella with **no children**, so "how close is Phase 0?" was unanswerable from the issue list. Every phase parent now carries a sub-issue tree, **closed issues included**, so each one renders a progress bar:

Phase parent	Children	Closed
#1 — Phase 0: research & reverse engineering	23	22 (#39 open)
#11 — Phase 1: interposer hardware (+ the standalone adapters #47, #56, #84)	8	2 (#13, #15)
#16 — Phase 2: firmware personalities	12	3 (#83, #86, #155)
#25 — Phase 3: Osborne-side software	4	0
#29 — Phase 4: on-hardware validation	0	—
#49 — Phase 5: the reproduction	3 (#52, #125, #128)	1 (#125)
#75 — the book	8 (#108–#115)	0

Two conventions came out of this, and both are worth keeping:

- **Closed work stays attached.** #2, #3, #4, #5, #6, #7, #8, #9, #10, #30, #40, #44, #64, #70, #71, #72, #73, #74, #79, #81, #95 and #168 are children of #1. The progress bar is the point: Phase 0 is **22 of 23** and should read that way.
- **`misc` is the label for work that is not a phase** — upstream contributions (#35, #82), the site (#97), build tooling (#106) and the modern font (#107). Everything else belongs to a phase, and to a parent.
- **`misc` is the label for work that is not a phase** — upstream contributions (#35, #82), the site (#97), build tooling (#106) and the modern font (#107). Everything else belongs to a phase, and to a parent.

The rule that keeps it honest: when an issue is opened, closed or re-scoped, the tree and this table move in the same PR — the same way a doc row does.

Appendix — decisions already taken

Kept here because they were reached by discussion and are not re-derivable from a source:

- **All Osborne 1s are NMOS.** The CMOS Z84C00 shipped in 1985; O1 production ended in 1983. The interposer uses 74AHCT/HCT; the replica uses a modern CMOS Z84C00 deliberately.
- **The interposer gets its own 5 VDC feed** near the Z80 socket (~300 mA+).
- **Bank switching cannot be latched.** `ROM MODE*` is re-derived on every instruction fetch; a bank-2 access is only meaningful while the CPU is actually in bank 2 (`o1-memory-io-map.md` §4).
- **The interposer must never drive I/O addresses `0x00–0x03`** — any `OUT` there toggles the bank latch. The ROM never does it, and there is no software bank switch to model.
- **The `52-`/`80-`/`104-` floppy prefix is a label, not a bootability gate.** The display width is the machine's; the text layout is the CBIOS's (`mame-emulation.md` §4).

Appendices

Archive provenance

This directory is the project's **research archive**: the documentation, disk images, and photographs that the reverse-engineering and design work under `docs/` and the GitHub issues depend on.

These files are reserved as safety copies of the documentation necessary to complete this project. They are committed in-repo on purpose: upstream archives move, go offline, or drop files, and this project cannot finish without these specific documents. If an upstream link dies, the in-repo copy is authoritative. Please do not "clean up" large PDFs or disk images from this tree — their presence here is the point.

Where each item was sourced is recorded below, so provenance is preserved alongside the copies.

OCR convention: most of the scans here have no text layer. When a document is OCR'd for research, the text is committed **adjacent to the PDF** as `<filename>.ocr.txt` (e.g. `drive_c/`

Drive_C_Users_Manual.ocr.txt) so it never has to be regenerated. Regenerate with tools/ocr_pdf.sh if a better pass is needed.

osborne1/ — Osborne 1 mainboard documentation

File	What it is	Sourced from
2F00153-01_Osborne1TechnicalManual_1982.pdf (19 MB)	Osborne 1 Technical Manual, 1982. No text layer — image-only scan. Theory of operations, banking, I/O, video, disk, keyboard.	bitsavers
2F00040_service_2ndEd_1983.pdf (12 MB, 292 pp)	**Osborne Field Service Manual, 2nd Edition, 1983** (part no. 2F00040-00) — the **whole-machine** service manual: board-level theory, parts lists, troubleshooting and the schematic set. The general-purpose companion to the Technical Manual above, and the source several docs/ pages already cite by part number. §6.12 documents the ScreenPac IC-harness fly-lead solder connections (used by firmware/screenpac-video/); it also backs docs/floppy-adapter-design.md (drive A's 150 Ω terminator, the P8 male header, DS1 strapping) and docs/o1-memory-io-map.md.	bitsavers
2F00040_service_2ndEd_1983.ocr.txt	OCR text layer (per the convention above), generated 2026-10-09	—

OCC1_1A2011-00_Schem_RevE.pdf (1.2 MB) | **Two drawing sets in one 12-page scan.** Pages 1–3: DISK ELECTRONICS, DWG 1A3004-00/01 Rev C (sheets 1–3 of 3 — the *drive's* own board). Pages 4–12: MAIN PC BOARD, DWG 1A2011-00 Rev E (sheets 1–9 of 9). 600 dpi bilevel; the embedded text layer is a useless early-90s OCR pass. Indexed, and the Z80 socket and P8 floppy pinouts extracted, in docs/o1-mainboard-schematic.md (#62). | bitsavers (also mirrored as Osborne_1_Schematics_1A2011-00_Rev_E.pdf on RetroTechCollection/Internet Archive) |

`OCC1_1A2011-00_Schem_RevE.ocr.txt` | OCR text layer for the above (per the convention below), generated 2026-10-10 at 400 dpi. Good enough to grep for component references; **not** reliable for pin-level detail — read sheets visually with `tools/schematic_render.sh`. | — |
`keyboard/osborne_keyboard_pinout.html` + `keyboard/PROVENANCE.md` | Verbatim capture of the pinouts.ru Osborne 1 keyboard page: P4 pinout (incl. pin 19 = +12V via J6+R21) and the full key matrix map. Used by #45/#47. | pinouts.ru, captured 2026-10-08 (provenance in the folder) |
`floppy-adapter/` (README, LICENSE, `.sch`, `.kicad_pcb`, PROVENANCE.md) | Snapshot of WayneVisser's Osborne1FloppyAdapter KiCad project — passive Gotek adapter documenting the O1′s "Shugart-like" floppy interface (power over the cable's unused pins). Used by #20/#49. | github.com/WayneVisser/Osborne1FloppyAdapter, captured 2026-10-09 (provenance in the folder) |
`modem/o1techman-interface-extract.pdf` (1 MB, 10 pp) | Chapter 4 "Osborne 1 Interface Design" extract of the O1 Technical Manual: §4.1 IEEE-488 (P3), §4.2 RS-232 (P2), **§4.3 MODEM (P1)**. Authoritative source for the MODEM DE-9P pinout — **TTL levels, not RS-232** — incl. pin 4 MSB, pin 5 CTS, pin 8 MCB, pin 9 RI, pin 7 +12V via 22Ω. Used by #56 (and #21/#12). **Also the P3 (IEEE-488) pinout**: §4.1.2′s 26-pin edge table, §4.1.1′s signal directions and §4.1.5′s Centronics re-mapping, transcribed in `docs/o1-ieee488-port.md` §6 (#84). Image-only scan — OCR it, `pdftotext` returns nothing. | kev.pulo.com.au/osborne1, captured 2026-10-09 |
`o1doubledensityusersguide.pdf` (2.2 MB, 15 pp) | *Osborne 1 Double Density User Manual*. The DD option's own manual, and the primary source for what it actually does: it is a **multi-density** option that recognises six formats, not just a denser one. §"formats automatically recognized" is quoted in `docs/dd-upgrade-formats.md` (#95). Image-only scan; OCR'd 2026-10-10 at 300 dpi as `o1doubledensityusersguide.ocr.txt`. | kev.pulo.com.au/osborne1, supplied by the maintainer 2026-10-10 |
`Comm-Pac_Users_Manual.pdf` (3.1 MB, 43 pp) | *Osborne Comm-Pac User's Manual* — **part no. 2F-00145-00**, "Osborne Datacom Modem + AMCALL Communications Program" (AMCALL is MicroCall Services, 1982; manual by Joseph Caggiano). The modem package's own manual: UART configuration, file transfer and its spec, Touch-Tone dialing, X-ON/X-OFF protocol, voice communication, diskette buffering, `UNLOAD.COM`, HELP menu. Used by #21 and #56. **Has a text layer** (Adobe Paper Capture, 2009), so it is greppable as-is and carries no `.ocr.txt`. | bitsavers `bits/Osborne/Osborne1/Comm-Pac/` (byte-identical) |
`o1astecpsac8151.pdf` (3.4 MB, 11 pp) | **"Repair Manual for Astec Power Supply AC8151"** — the Osborne 1′s own PSU. Test set-up (isolation transformer ≥500 VA, 0–28 V variac, DMMs, scope with X10/X100 probes, load board), the 115/230 V input-select wire, and a performance table for the +5/+12/−12 rails at 190/220/270 VAC. Image-only scan; OCR'd 2026-10-14 at 300 dpi as `o1astecpsac8151.ocr.txt`. Serves #52. | kev.pulo.com.au/osborne1 (byte-identical) |
`o1astecpsac8151.ocr.txt` | OCR text layer for the above (per the convention above), generated 2026-10-14 at 300 dpi | — |
`o1md1000-190specs.pdf` (4.3 MB, 11 pp) | **Model MD1000-190 specifications (preliminary)** — the stock Osborne 1 **5-inch monitor**: 127 mm diagonal, 3.80 × 2.54 in display, 55° deflection, 650 lines centre / 500 corner, 10 Hz–12 MHz bandwidth, 15.75 kHz horizontal and 50/60 Hz vertical sync, TTL sync drive, positive-white video, **+12 VDC ±0.5 V**. Also carries service notes (CRT 2nd-anode discharge) and a schematic. Image-only scan; OCR'd 2026-10-14 at 300 dpi as

o1md1000-190specs.ocr.txt. Serves #24 and the mechanical work. | kev.pulo.com.au/osborne1 (byte-identical) |
o1md1000-190specs.ocr.txt | OCR text layer for the above (per the convention above), generated 2026-10-14 at 300 dpi | — |

Basis for docs/o1-memory-io-map.md (#40) and docs/o1-mainboard-schematic.md (#62). Both scans are image-only; extraction is manual — OCR for text pages, and for schematics a native-resolution region render (tools/schematic_render.sh) read by eye. The mainboard map is independently cross-checked against the running machine in MAME; see docs/o1-mainboard-schematic.md §7 (#64).

Two further scans offered on 2026-10-14 are deliberately **not** here, because the archive already holds them: o1screenpacusersguide.pdf is byte-identical to screenpac/o1screenpacusersguide.pdf, and 2F00040-00_Service2ndEdition_1983.pdf is a 4.4 MB re-compression of screenpac/2F00040_service_2ndEd_1983.pdf — which is the pristine 12 MB bitsavers copy and carries the OCR sidecar, so it is the better of the two. See #245.

inspired-by/ — adjacent projects

inspired-by/README.md — other Osborne 1 reinventions (MAME cyberdeck, Raspberry Pi brain transplant, Loxley's Gotek mod): what they did, and what we take from each. Contrast defines the project: **we keep the original machine intact**.

mx80/ — Epson MX-80 printer references

File	What it is	Sourced from
mx80/Epson_MX-80_Operations_Manual.pdf	Epson MX-80 operation manual; Appendix 4 (PDF p. 97+) has the character-font tables (6×9 matrix) and ESC code reference.	Internet Archive / manualsbase, captured 2026-10-09
mx80/EPSON_MX-80_Fonts_v1.0.zip + mx80/fonts/	Michael Walden's MX-80 font pack (CC BY-NC-SA 4.0). Convenience cross-check; see the folder README for the double-strike/2×-width caveat.	mw.rat.bz/MX-80, captured 2026-10-09

Also: drive_c/Drive_C_Users_Manual.ocr.txt — OCR text layer for the image-only Drive C manual scan (generated 2026-10-09 per the OCR convention above; source of the TurboPac details used by firmware/drive-c-gpib/).

References for the virtual Epson MX-80 printer personality (#19).

screenpac/ — OCC ScreenPac (80- and 104-column upgrade)

File	What it is	Sourced from
`o1screenpacusersguide.pdf` (2.3 MB, 20 pp)	*Osborne 1 SCREEN-PAC User Guide*. **The primary source for what the upgrade does**, and it is more than a wider window: p. 13 (PDF p. 17) gives the three software-selectable widths (52 / 80 / 104), horizontal scrolling in **32-column blocks**, and the **presence flag at `2400H` bank 2** that lets software detect the board. Quoted in `docs/mame-emulation.md` §2 and `docs/o1-memory-io-map.md` §2a/§3 (#73, #24).	kev.pulo.com.au/osborne1, supplied by the maintainer 2026-10-10

The ScreenPac is the mechanical/electrical template for the interposer and for the video-output (HDMI) personality (#24, #9 photos).

The **Field Service Manual** those ScreenPac notes come from (`2F00040`, §6.12) is a whole-machine document, so it is filed with the rest of them in `osborne1/` above — not here.

technical-articles/ — extracts from Osborne-era magazines

Only the pages that matter are kept, cut from the original and OCR'd. The full issues are not in the repo: they are 50–60 MB scans each, they are on the Internet Archive (https://archive.org/details/theportablecompanion) and on the maintainer's own backup, and keeping whole magazines for a few pages of schematics is not worth the repo weight. Each extract names the issue and the pages it was cut from, so it can be checked against the original.

Extract	What it is	Cut from
`TPC-Issue11-1983-pp32-osmosis-product-line-ad.pdf` (0.6 MB, 1 p)	OSMOSIS' full-page advertisement, "The Next Step Forward" — the whole product line **with product numbers**: **OSMOS 1** DD 184k/drive ($195), **1.5** ($335), **2** DS/DD 370k ($895), **3** DS/DD double-tracking 750k/1.5MB ($1050), **4** "read, write and format **over 20 other manufactures CP/M formats**" ($225), **5** the internal 80-column board ($250 — "**includes a Hard Disk interface for future plug-in expansion**", normal/inverted video, underline, ⅔ and ½ intensity), **6** telex, **7** Winchester to 42 MB, **8** instrumentation, **10** Olivetti ET121 interface; bundle OSMOS 1+4+5 for $385. Quoted in `docs/o1-upgrade-boards.md` and `docs/dd-upgrade-formats.md`. Relevant to #24, #11/#12 and #95/#30.	*The Portable Companion* #11, October 1983 — PDF p. 32 (printed 30)
`TPC-Issue11-1983-pp54-doubling-hard-disk-memory.pdf` (0.7 MB, 1 p)	"**Doubling hard disk memory**" by **Bob Gras** (reprinted from the OSUG newsletter, Anchorage) — patching Corvus's `CLINK2K2.ASM` so the link program works after a double-density upgrade: `DELTA EQU 500H` → `900H`, because "**more memory is required for managing the Osborne floppy drives**" under DD. The same page carries the tail of John Dilday's WordStar printer-spooler article.	*The Portable Companion* #11, October 1983 — PDF p. 54 (printed 52)
`TPC-Issue11-1983-pp41-45-[illegible].pdf` (0.9 MB, 5 p)	[illegible]	*The Portable Companion* #11, October 1983 — PDF pp. 41–45 (printed 39–43)

How these extracts are made

The maintainer reads the issues and hands over the ones worth keeping, giving the **PDF page sequence** to cut. The standing workflow:

1. Cut the pages: `pdfseparate -f A -l B ISSUE.pdf /tmp/p%03d.pdf` then `pdfunite /tmp/p*.pdf OUT.pdf`.
2. OCR the extract with `tools/ocr_pdf.sh OUT.pdf` — adjacent, per the convention above. **Wait for its `Wrote <path>` line before committing**; the file exists while it is still being written.
   - **If you drive `tesseract` by hand, do not render into `/tmp`.** On this machine it cannot open images under `/tmp` (Leptonica reports the file as not found) and, with `2>/dev/null`, fails *silently* — a 43-page sweep over `/tmp` returned nothing at all and looked like "no matches". The script is fine because it renders into a `mktemp -d`.
   - **A display-type page can defeat tesseract entirely.** Full-page advertisements set in display faces over tinted screens sometimes OCR to nothing but the folio — the #5 Mega-byte MD-10 ad (`TPC-Issue05-1983-pp25-…`) was the first. Do **not** commit an empty `.ocr.txt`: transcribe the page by eye into the file behind a `[HAND TRANSCRIPTION, not tesseract output …]` header, so the text still lives beside the scan and never has to be re-derived.
3. Add a row below, and write the findings into the `docs/` page they bear on with the extract cited.

Two things to know before cutting pages:

- **Page numbers are PDF pages, not printed pages.** Every page checked so far prints its own number two lower (`PDF = printed + 2`) — issues 11, 13, 14, 15, and now #5 p. 25 (⟶ 23) and #7 pp. 50 and 56 (⟶ 48 and 54). So an article "on page 26" is PDF page 26 and printed page 24 — and cutting printed 26 would miss the article's title page. **The offset is not uniform, though:** the #5 extract filed as `pp98` carries folio **84**, i.e. `+14`, so an issue can contain unnumbered pages. Read the folio off the page rather than assuming — `pdftotext -f N -l N -layout ISSUE.pdf - | tail -3` prints it, and the extract's own `.ocr.txt` usually ends with it.
- **Provenance for every article from *The Portable Companion* is the same collection**: https://archive.org/details/theportablecompanion. Rows therefore cite the issue and the page range rather than a per-file URL.

The **full issues are not kept in the repo** — 50–60 MB each, and they live on the Internet Archive and on the maintainer's own backup. Only the extracts are committed, and they are cut from files outside the repo (the maintainer's `~/Documents` tree), never from a copy inside it.

copower88/ — SWP CoPower-88 (Z80-coprocessor board being reverse-engineered)

Path	What it is	Sourced from
`zorba/Co-Power-88guide.pdf`, `zorba/Co-Power-88advert.pdf`, `zorba/611-0003-1..4.pdf`	SWP CoPower-88 user guide, advertisement, and SWP technical document scans (Zorba-flavored set)	zorba.z80.de — the Zorba Portable Computer archive site
`zorba/*.td0`, `kaypro/*.td0`	TeleDisk images of CoPower-88 CP/M-86 / DOS boot disks (Zorba and Kaypro versions)	zorba.z80.de (exact path not recorded in repo history)
`zorba/extracted/`, `kaypro/extracted_cpm/`, `*/disasm/`	CP/M files extracted from those images by the project pipeline (#2), and disassemblies of the CoPower RAM-disk/DOS drivers	Derived in-project — not external sources
`swp/Copower_User_guide.pdf`, `swp/Swp_ad.jpg`	Fuller CoPower-88 user guide (4.7 MB, not the Zorba set) and an SWP advert scan	kayprojournal.com `Swp_copower_88` — `File:Copower User guide.pdf`, `File:Swp ad.jpg`
`swp/1600px-73356_8088_card.jpg`, `swp/1600px-73356_mem_card.jpg`, `swp/866px-73356_cpu_card.jpg`	Board photographs of the CoPower-88 8088 card, memory card and CPU card — held as MediaWiki thumbnails (1600 px / 866 px) of `File:73356 8088 card.jpg`, `File:73356 mem card.jpg`, `File:73356 cpu card.jpg`	Same page: kayprojournal.com `Swp_copower_88`

The whole `swp/` set is from that one page. Confirmed 2026-10-10 by querying the wiki's API (`list=allimages` and `prop=images&titles=Swp copower 88`) and matching all five `File:` names exactly — not by reading the rendered page text. The wiki also carries `File:Copower sch 1..4.pdf`, a second copy of the SWP schematics we hold from zorba.z80.de.

Basis for #3/#4/#5 (CoPower-88 protocol analysis). The board's own 8088 ROM is `roms/swp-p88.rom` — see `roms/` below and #70.

drive_c/ — Drive C (IEEE-488 ramdisk, #6)

Path	What it is	Sourced from
`Drive_C_Users_Manual.pdf`	Drive C user manual (IEEE-488 virtual disk drive), **second edition June 1984**	bitsavers `bits/Osborne/Osborne1/Drive_C/` (upstream name `Drive_C_-_Users_Manual.pdf`)
`Drive_C_Executive_Users_Manual.pdf`	**Drive C user manual for Osborne Executive computers** — the same product on the other host: identical hardware appendix, `ECL`/`ECN`/`ECU` software under CP/M+ instead of `DCL`/`DCN`/`DCU` under CP/M 2.2, and **no `DFD.SPR`**. First edition July 1984, second printing 9/84; cover stamped PRELIMINARY, running headers "final draft". 111 pp, 12 MB, image-only (150 dpi grayscale re-encode — see the note below). Analysis: `docs/drive-c-executive.md` (#84, #6, #145)	**Jim Rayburn**, private Facebook group, posted 2024-01-06 — posts …/2063148960719887 and …/2023247321376718; supplied to the maintainer as `Executive_DriveC.pdf`, 2026-10-11
`Drive_C_Executive_Users_Manual.ocr.txt`	OCR text layer for the above — 300 dpi `tesseract`, per the convention below	Derived in-project, 2026-10-11
`DRIVE_C.IMD`	ImageDisk image of the Drive C utility disk (label comment `Drive C for the Osborne 1 computer`)	same folder — **not** imaged in-project; corrected 2026-10-10
`Drive_C_disk_label.jpg`	Photo of the physical disk label	same folder (15 044 B, byte-identical) — **not** a project photo; corrected 2026-10-10
`extracted/`	Files extracted from `DRIVE_C.IMD` by the project pipeline (#2)	Derived in-project
`disasm/`	`dcl.asm` + `dfd.asm` — z80dasm listings of the installer and the resident driver, for the protocol work (#6). Regenerate with the exact command in `docs/drive-c-protocol.md` §Disassembling	Derived in-project

A note on the Executive manual's size. The owner supplied a 300 dpi scan, 42.33 MB — larger than any other single file in this tree. It is committed as a **150 dpi re-encode** instead (12.13 MB), the same resolution as the Drive C manual above: the document is a typewritten mono manual, and the page images carry **no colour at all** — every pixel is neutral grey, checked across all 111 pages — so the extra resolution bought nothing but file size. All **111 pages** and the page geometry are preserved. The **300 dpi text layer** was taken from the owner's original before the re-encode and sits beside the PDF as `Drive_C_Executive_Users_Manual.ocr.txt`, so no text was lost. It is recorded as a re-encode rather than as the owner's original, so the provenance stays honest: the file here is **not** byte-for-byte what Jim Rayburn posted.

rtc/ — RT-60A real-time clock (#7)

Path	What it is	Sourced from
`RT-60A_Manual.pdf`	RT-60A RTC manual (JG Communications)	bitsavers `bits/Osborne/Osborne1/Osborne_1_Real_Time_Clock/` (upstream name `RT-60A_Real_Time_Clock_Manual_-_Osborne_1.pdf`)
`RT-60A.TD0`	TeleDisk image of the RT-60A utility disk — **a damaged dump** (27 no-id sectors); filesystem extraction fails, see `tools/README.md`	same folder

harddisk/ — ACT/OCC1 hard disk (#8)

Path	What it is	Sourced from
`OCC1_HARDDISK.IMD`	ImageDisk image of the OCC1 hard-disk system disk (label comment `act-osborne / hard disk software / rev 2`)	bitsavers `bits/Osborne/Osborne1/_floppy_images/` — **not** imaged in-project; corrected 2026-10-10
`extracted/`	Files extracted from the image, incl. `hardbios.hex`	Derived in-project

adaptec/ — Adaptec ACB-4000 hard-disk software (2026-10)

Raw Osborne 1 SSSD floppy images (102 400 B = 40 tracks × 10 sectors × 256 B, **no IMD header**) each with a photo of the disk's label. Source: Internet Archive, `md-utilities-adaptec-4000-controller-ver.-1.93-osborne`.

File	What it is
`MD Utilities Adaptec 4000 Controller Ver. 1.93 Osborne.img`	Adaptec ACB-4000 utilities
`MD Utilities Adaptec 4000 Controller Ver. 1.93 Osborne - MD-10 MD-20.img`	Same, for the MD-10/MD-20 drive pair
`Osborne Hard Disk Control Software Version 2.01 (1984).img`	Osborne's own hard-disk control software
`compatible-drives-1991-usenet.txt` (23 KB)	**A period list of ST-506/MFM drives**, not Osborne software: Robert L. Haar's *"List of replacement hard drives for 3b1"* (`comp.sys.3b1`, 12 Feb 1991, TUHS Usenet archive), which reposts Lenny Tropiano's *"ST-506 MFM Hard Drive Manufacturer listing (LONG), model #'s and specs"* (`unix-pc.general/comp.sys.att/comp.periphs`, 2 Sep 89), itself edited out of the full drive list on the Western Digital technical-support BBS. Per manufacturer: model, formatted capacity, cylinders, heads, reduced-write-current, write-precomp, encoding, transfer rate, access time and **sectors/track** — the last of which is **17** for almost every entry, where `DRIVETBL.DAT`'s capacity formula uses 18, and it gives `PYXIS-13` **320** cylinders where `DRIVETBL.DAT` says **321**. Archived because the personality should offer any ST-506/412-class drive, not only the 126 the MD-10 tools ship (`docs/harddisk-decision.md` §7.2)

Together with `acb4000a.rom` (Maslin's archive) this identifies the controller behind `harddisk/OCC1_HARDDISK.IMD` — issue #8.

adaptec/md10-system/ — the MD-10 operating system (2026-10, generated in-project)

The system the archive did not hold. `docs/harddisk-decision.md` §7.3 concluded that nothing in the archive initialises the MD-10's low-RAM tables and that the utilities disk carries no system image — which left #179's acceptance ("boots CP/M from the emulated hard disk") with nothing to boot. It was not obtainable by *reading*; it is obtainable by *running*, and the vendor's own installer produces it: `PREP.COM` option 3, "Configure operating system", writes it to the drive. Captured off configured drives on 2026-10-12 and 2026-10-14.

File	What it is
`lba0-15.bin` (8 KiB)	**PYXIS13**, two partitions of 11544 blocks: the first sixteen blocks of a configured drive — LBA 0 the disk label and its partition table, LBA 1–3 the header, LBA 4 a table, LBA 5–15 the operating system
`md10-os.bin` (5.6 KiB)	its operating system — LBA 5, 11 blocks (0x1600 bytes) — relocated 0x100 high, so it needs `tools/research/md10reloc.py` before it boots (#198)
`lba0-15-p27.bin` (8 KiB)	**PYXIS27**, **three** partitions of 15399 blocks — the driver's own static fallback table, and the only layout that matches its shared DPB. **The floor pin**
`md10-os-p27.bin` (5.6 KiB)	its operating system, already at its load base (`JP 0C4C8h`, with `LD SP,nn` at the target) — no relocation step
`README.md`	provenance, and what each file is

Derived in-project from `adaptec/*.img` (the `PREP.COM` that writes it) and the 2.01 utilities disk (the `HARDCCP.SPR`/`HARDBDOS.SPR` it writes); the PYXIS13 drive it was taken from had md5 `7fa2a3836813dabdeaaa664ea88d002e`, and `tools/research/md10drive.py` reproduces **either** drive byte for byte from these files (`--layout md10` / `--layout md20`). The PYXIS27 pair was produced by `tools/research/md10prep.sh 321,8,18 3` and independently reproduced by it. The recipe, the MD-10's memory map, the boot chain and the one-DPB constraint are in `docs/md10-system.md`.

adaptec/acb4000/ — Adaptec ACB-4000 controller hardware (2026-10)
The controller itself — schematics, board photographs, the controller's own firmware ROMs, and two OEM manuals. Archived because #8 **selected this controller** (`docs/harddisk-decision.md`). Source: bitsavers `pdf/adaptec/ACB-4000/`.

Path	What it is
`500504-00_ACB-4000_OEM_Manual_Mar84.pdf` (9.8 MB)	OEM manual — the command set (Table 6-1), the reference host-adapter register model, a sample Z80 driver and a sample format sequence
`400003-00A_ACB-4000A_Users_Manual_Oct85.pdf` (7.1 MB)	ACB-4000A user manual. **Image-only scan** — `pdftotext` returns nothing. Carries **§2.3.1** (device independence / auto-configuration) and **§3.7** (power-up operation): the drive parameters are encoded onto the disk at `FORMAT`, **in the ID field of cylinder 6**, and read back at power-up — see `docs/harddisk-decision.md` §7.3
`400003-00A_ACB-4000A_Users_Manual_Oct85.ocr.txt` (177 KB)	OCR text layer for the above (per the convention at the top of this file), generated 2026-10-12 at 120 dpi with `tools/ocr_pdf.sh`. Good enough to grep — §2.3.1's "ID field of cylinder 6" is legible — but the OCR mangles part numbers (`ACB-4000` renders as `ACB-4909`/`ACB-4990` in places), so read the PDF for anything load-bearing
`ACB_4000_and_5000_Disk_Controllers_OEM_Manual_Preliminary_May83.pdf` (2.6 MB)	preliminary OEM manual (ACB-4000 + ACB-5000)
`ACB4000A_Schematic_ARD_2025.pdf` (2.4 MB)	board schematic
`firmware/`	six 8 KiB controller ROM images + an Intel-HEX build (`ACB-4000A-rev-B.hex`)
`pictures/`	seven board photographs

The `adaptec/*.img` software above is the **host** side (the MD-10 driver); this folder is the **controller** side. The personality needs both: the host side is what it must answer, the controller side is what it must be.

diagnostics/ — Osborne 1 confidence tests (2026-10)

File	Label comment
`DIAG_2.04.IMD`	`2.04 / diagnostics`
`OCC1_DIAG_2.1.IMD`	`occ diag / ver 2.1 / 12/31/84`

Source: bitsavers `bits/Osborne/Osborne1/_floppy_images/`. These are the known-good boot images the validation matrix (#29) needs.

nuevo/ — Nuevo Electronics double-density upgrade (2026-10)

File	What it is	Sourced from
`Nuevo_Electronics_DD_Upgrade_Manual.pdf` (13 MB)	DD upgrade manual	bitsavers `bits/Osborne/Osborne1/` (13 810 755 B, exact match)
`Nuevo_Electronics_DD_Upgrade_Manual.ocr.txt` (919 lines)	OCR of the manual — the BIOS 1.4⟶1.5 difference list (DPBs for IBM/Xerox/DEC removed, 80-column presence check, jump table at `E100h`), the 380K double-sided claim and the single/double-density system-track rules, all quoted in `docs/dd-upgrade-formats.md`	derived in-project with `tools/ocr_pdf.sh … 300`
`OS1NUEVO.IMD`	`Osborne 1 with DD and 80 col mods` boot disk, reimaged 2026-10-10 with DiskUtilities	derived in-project
`photos/nuevo-80-column-board.jpg`	**80-column board, `REV. D`, serial 4348** — the first hardware view of the product. Its own **NEC D780C-1** (Z80), a Nuevo **`SY6545A-1`** CRTC, 2 × `SRM2016C25` SRAM, a socketed **`CHARACTER GENERATOR`** ROM dated **6-29-84**, a 12.288 MHz oscillator, and the optional `BAUD` generator. Devices, and what they imply, in `docs/o1-upgrade-boards.md` §"The Nuevo boards themselves" (#73, #24)	**Mike Loewen**, private Facebook group, 2024-10-14 — post
`photos/nuevo-double-density-board.jpg`	**DD board, serial 20011** — `DAL DENS` silk, a 34-pin IDC header with a `STD DRIVE` jumper, and a discrete write-precompensation/separator chain (`74LS195A`, `SN74LS14N` ×2, `SN74LS00N` ×2, `SN74LS02N`) plus a 16-pin `9216`-family device (U1). Same analysis as above	same post
`photos/nuevo-80-column-and-double-density-boards.jpg`	Both boards in one frame — the shot the 2024-10-14 post is built around	same post

The photographs are **third-party** (Mike Loewen, a private group) rather than in-project like `pictures/`, and they are the only hardware views of either Nuevo product — there is no Nuevo schematic in the archive. Neither board bears on Drive C: both are host-side (video, and floppy encoding).

> **Removed 2026-10-10: `NUEV0151.BIN`.** It was a byte-identical duplicate of `tools/emulator-setup/roms/monrom-rev1.51-12.ud11` — same 4096 bytes, SHA-256 `aec8f5be127fdff5d30cae0b7f45ee48d13a8c1d5745a262b944b9ba67778906`, CRC32 `298da402` — verified with `cmp` before deletion. It carried no provenance of its own, so the tracked emulator-set copy, which has a recorded source, is the one we keep. The BIOS strings are unchanged by this: `COPYRIGHT 1983, OSBORNE COMPUTER CORP. / COPYRIGHT 1984, NUEVO ELECTRONICS CORP.`

Nuevo shipped **two** products — a DD upgrade and an 80-column board — and neither matches Osborne's own implementation of that function; see #73.

`OS1NUEVO.IMD` **is** the Nuevo 80-column system disk: its system area carries `CBIOS vers 1.5` and the text *"THIS IS AN EIGHTY (80) COLUMN DISPLAY FOR THE OSBORNE ONE"*. It does **not** currently boot under MAME — it prints the CBIOS banner and stops (measured over 260 emulated seconds; the pristine upstream copy behaves the same), which blocks validating the Nuevo video path in the emulator. See #79 and `docs/mame-emulation.md` §4.

osmosis/ — Osmosis Computer upgrades (2026-10)

Source: bitsavers `bits/Osborne/Osborne1/Osmosis_Upgrades/` — all seven files byte-for-byte identical.

`Osmosis_CPM_Disk_Emulator.pdf`, `Osmosis_80_Column_Board.pdf`, `Osmosis_SSDD_Upgrade.pdf`, `OSMO-EMU.TD0`, `OSMOS-DD.TD0`, plus two disk-label photos. Osmosis likewise shipped two products (a CPM disk emulator and an 80-column board). **The 1983 CPM Disk Emulation System is direct prior art for this project** — see #73 and `inspired-by/`.

gotek/FF/ — working Gotek configuration (2026-10)

`FF.CFG` + `IMG.CFG` — the project's own Gotek configuration for the O1: `interface = shugart`, `pin02`/`pin34` = `nc`, and an `IMG.CFG` default of 40 cylinders × 1 head × 16 × 128 B, FM, 125 kbit/s — i.e. O1 SSSD. Captured from the physical Gotek, so these are primary sources, not downloads. Used by #62 and #20.

roms/ — ROM images (2026-10)

Canonical inventory, CRC32s and the third-party ROM survey are tracked in #70. Files here come from two sources:

- `os1-143.rom`, `os1-144.rom`, `os1-vid8.rom`, `os1vid80.rom`, `swp-p88.rom` — Don Maslin's ROM archive on retroarchive.org
- `OZROM-1E/OZROM_1E.BIN`, `OZROM-1E/OZROM_1E_Manual.pdf`, `OCC1_ScreenPac_RevA.BIN` — github.com/BrettHallen/Osborne_1 `ROM/`

`swp-p88.rom` is **8088 code, not Z80**: the SWP co-processor board's own ROM. The board and interposer card are common to every host machine (only the driver varies), so this is the 8088-side firmware of the CoPower-88 family — #3/#4/#5.

`OZROM-1E/OZROM_1E_Manual.ocr.txt` — OCR text layer for the OZROM manual (image-only scan; generated 2026-10-10 at 300 dpi with `tesseract`, per the OCR convention above). Good enough to grep for the technical appendices — which is how the IEEE-488 removal and the 3 KB video window were found (#81) — but not reliable for exact code or pin detail.

`OZROM-1E/OZROM_1E_Utilities.IMD` — the **OZROM 1E Utilities disk**, the software half of the OZROM package, which we did not previously have. ImageDisk 1.19, label `ozrom`; 40 cyl × 10 × 256 B FM (O1 single density), 22 of 40 tracks in use. Its CP/M directory is complete and readable: 20 files, all Micro Management 1984, with `READ.ME`, `CRCKLIST.CRC` and the docs for `TIMER`, `CLKPATCH`, `DVORAK` and `TD`. **Condition: whole — all twenty files complete.** 290 of the 400 sectors read back as uniform `0xE5`, and that is CP/M **format fill** (free blocks), not damage: every file extracts at the size its directory entry gives, no file holds an interior run of `0xE5` longer than a byte, and `READ.ME` reads as continuous prose across all five of its sector boundaries. The "most files are truncated or holed" reading this row used to carry was our own sector-skew bug (#170), and it is what a search for a complete copy was filed on (#168, since closed). It is a data disk, not a system disk — the boot tracks carry a stub whose message is "There is no CP/M system on this disk." Provenance: supplied by the project owner as a local file (2026-10-11); the original source URL is not yet recorded, so treat the chain of custody as open. Analysis: `docs/ozrom-1e.md` §10.2 and §10.5. Used by #81.

shipped-software/ — the shipped application library (2026-10)
Fourteen ImageDisk images of the disks that shipped with (or were published for) the Osborne 1. The ImageDisk label comment is quoted because it is the disk's own identity. Source: **Dave Dunfield's ImageDisk archive** (`dunfield.classiccmp.org`, which rotates its directory to discourage deep links) — `d/osborne1.zip` and `d/o1ddsys.zip`, byte-identical to these copies.

File	Label comment
O1CPM.IMD	Osborne-1 CP/M System Disk / Double Density
O1CBMB.IMD	Osborne 1 - CBASIC/MBASIC / Double Density
O1SCALC.IMD	Osborne 1 - SuperCalc 1.12 / Double Density
O1WSMM.IMD	Osborne 1 - WordStar/MailMerge 2.26 / Double Density
OS1SYSD.IMD	CP/M-2.2 system disk for Osborne 1 w/ DD drives / SSDD 1024 byte sector
OS1SYSS.IMD	CP/M 2.2 System Disk for Osborne 1 / SSSD 256 byte sector
OS1BASIC.IMD	CP/M 2.2 Basic Disk for Osborne 1
OS1DBASE.IMD	dBase II for the Osborne 1 on sysgened disk
OS1DIAS.IMD	CP/M 2.2 Diagnostics Disk for Osborne 1
OS1MCAL.IMD	CP/M 2.2 Mocro-Call Disk for Osborne 1 (Communications)
OS1MDM7.IMD	Modem 740 communications for Osborne 1
OS1UTLS.IMD	CP/M 2.2 Utilities Disk for Osborne 1
OS1XUTLS.IMD	Extended utilities for Osborne 1 w/ 80 column video card
OS1WRDST.IMD	CP/M 2.2 WordStar 3.0 Disk for Osborne 1

The corpus for the image library (#20), the updated boot disk (#28) and the validation matrix (#29). Dunfield also publishes a 2005 single-density set (`O1CPMS.IMD`, `O1CPMU.IMD`, …) which is deliberately **not** copied here.

source-code/ — Osborne's own firmware source (2026-10)

Path	What it is	Sourced from
`BIOS-144-source/` (18 files)	**The original Osborne ROM source**: `rom144.asm` (DOUBLE DENSITY ROM REV 1.44, Roger W. Chapman, 2/4/1983), `occbio05..95.asm` (OCC CBIOS Rev 1.41), `occram15/25.asm`, `occtxt6.ast` (ACT80 assembler equates), `bios141.gen`, `release.txt`, `tran.txt`. Basis for #72.	github.com/BrettHallen/Osborne_1 `ROM/Source_Code_v144/` (all 18 file sizes match)
`ROM-13-source/` (21 files)	**The ROM 1.3 source, extracted from the disk below** (#205): `occrom03.asm` (the master, which `LINK`s the rest), `occrom13..93.asm`, `occbio03..83.asm` (CBIOS rev 3), plus `occram2.asm` and `occtxt6.ast`, which are **not** on the ROM disk and come from `OSUTLSRC.IMD`. The 8-inch disk is 77 tracks, track 0 FM 26×128, tracks 1–76 MFM 8×1024, CP/M 2.2 with **skew 3**; read with `tools/research/o1imd8.py`. See `docs/rom-13-reconstruction.md`.	`OSROM13.IMD` + `OSUTLSRC.IMD` below
`BIOS-200-source/` (3 files)	**ROM 2.0′s monitor source** (#247): `rom2.diff`, the change to `BIOS-144-source/rom144.asm` as a unified diff — `HCBOOT`, the two re-lettered boot keys, the boot screen and `IMSG`'s own slot — with `rom2-screen.asm` and the folder's own README. There is deliberately **no** `rom2.asm`: a second copy of a 4,655-line vendor file would drift from the first silently, and the point of this monitor is that it is the shipped 1.44 with new code in the free tail. `tools/research/rom2asm.sh` applies the diff, assembles with the vendor's	in-project (#247)
`OCCCBIOS.IMD`, `OCCROM.IMD`, `OCCUTIL.IMD`, `OSBIOSSR.IMD`, `OSBROM.IMD`, `OSROM13.IMD`, `OSUTLSRC.IMD`	OCC engineering **source disks** (1981–82): CBIOS, ROM rev B, utility programs, ROM 1.3. The ImageDisk comments carry the OCC part numbers	bitsavers

Licensing: `release.txt` is Osborne's 1985 letter permitting the First Osborne Group to reprint the schematics and the BIOS listing, on condition that any money goes to the non-profit FOG and that the trademark/copyright notices are retained. FOG is long defunct and this material is already public in several places — this copy came from a public GitHub repository. The notice is retained verbatim; if the terms ever need re-examination, that is the file.

act-assembler/ — SORCIM ACT80, the assembler the firmware source needs (2026-10)
The Osborne firmware source in `source-code/` is written for **SORCIM's ACT** ("Assembly Code Translator"), not for any assembler we had. This is that assembler, kept as a safety copy because `rom144.asm` reassembles with it to the shipped ROM **byte for byte** (#203, `docs/o1-rom-source.md`).

Path	What it is	Sourced from
`ACT80.COM` (13 312 B) md5 `4f76a911a66ddc8e1e02a1676c232d55`	**SORCIM ACT80, version 3.0** — 8080/8085 + Z80. The build the 1.44 reassembly was done with.	cpmarchives.classiccmp.org `cpm/Software/rlee/S/SORCIM/ACT Cross Assemblers/`
`3.2/ACT80.COM` (13 312 B) md5 `cf73590466736fd1960e829b5e221aa0`	**SORCIM ACT80, version 3.2**, serial 279. Held so the version question can be re-checked against a local file rather than the mirror: it assembles `rom144.asm` to output **byte-identical** to 3.0's.	same mirror, `cpm/Software/bbe/SECPM/119/` — `073` and `087` hold the same bytes
`3.2/ACT80-eca-bundle.COM` (13 312 B) md5 `88256731a9399dee68a29054865d072c`	A **third, different 3.2 build**, the one that circulated inside a third-party software bundle. It reports `SORCIM 808x Assembler ver 3.2` in its listing banner, but carries **no SORCIM copyright or serial line at all**, and it differs from the SORCIM 3.2 in **13 112 of 13 312 bytes** — a redistribution rather than the release, kept as evidence of how many binaries claim to be 3.2.	same mirror, `cpm/Software/rlee/D/DIGITAL RESEARCH/ECA/` (it is named `ACT80.COM` there; renamed here so both 3.2 copies can sit side by side)
`ACT65.COM`, `ACT68.COM`, `DOCOP65.ASM`, `DOCOP68.ASM`	The same kit's 6502 and 6800 translators and their mnemonic listings. Kept with the kit rather than dropped — they are 30 KB, and a hand-picked subset invites the question of what was left out.	same
`DOCOP80.ASM` (8 192 B)	**Every ACT mnemonic, in alphabetical order** — the dialect reference, and the file to diff against if a future source is rejected.	same
`IPARAMS.ACD`, `INSTALL.ACD`, `ACDEG.ASM`	The kit's own installer, and its example program using the `ACT.LIB` include files, and the `SUBMIT` script for driving them (`LINK IPARAMS.acd`).	same

Versions. The mirror holds nine copies of `ACT80.COM` in **three distinct builds, and all three are kept here** — so the version question can be re-checked against a local file rather than the mirror:

Build	md5	Copies in the mirror	In-repo
3.0	`4f76a911…`	`S/SORCIM/ACT Cross Assemblers/`, `M/MORROW/OUTSIDE SOFTWARE/ACT/`, `L/LOOSECPM/{079,109,262}/`	the kit above
3.2, serial 279	`cf735904…`	`L/LOOSECPM/{073,087,119}/`	`3.2/ACT80.COM`
3.2, no banner	`88256731…`	`D/DIGITAL RESEARCH/ECA/`	`3.2/ACT80-eca-bundle.COM`

3.5e — the version `bios141.gen` names — is not in the mirror, and it does not need to be, because the version does not change the answer: **3.0 and the SORCIM 3.2 assemble `rom144.asm` to byte-identical output**, both re-checked against the in-repo copies (#203). Either one reproduces the shipped ROM:

```
01_ACT80=research/act-assembler/3.2/ACT80.COM tools/research/rom144-reassemble.sh
```

The third build is a caution rather than a spare. On the same disk, with the same command, it returns to the `A>` prompt immediately and prints **nothing** — no banner, no `no ERRORs` line, no hex file — where both SORCIM builds assemble the source cleanly. Whatever it is (a relocation for one system size, a bundle installer's variant, or a patched copy), it is **not a drop-in ACT80**, and it is kept as evidence of how many different binaries claim to be version 3.2.

Licensing: `READ.ME` is SORCIM's own 1980 notice — *"This disk contains software proprietary to SORCIM. It may NOT be copied other than for use by the purchaser."* It is retained verbatim and recorded here rather than quietly dropped. The kit has been publicly mirrored since 2011, and this copy exists to read the firmware source the project already holds; if the terms ever need re-examining, that file is where to start.

`boot-screen/` — the authored ROM 2.00 boot-screen art (2026-10, in-project)

Not a sourced document: **the maintainer's own drawing** of ROM 2.00′s boot screen, authored in the machine's graphics characters. A design source for #83, kept here for the same reason `pictures/` is — it exists nowhere else.

File	What it is	Sourced from
`boot-screen/o1next-art-1ng.txt`	the **1NG** art: 125 × 28 cells, one character per cell, SHA-256 `a92bf28e81b54c58f59c48ab16f090e2d26c4e8d3705ed0cfcda7dd10bb1f9ba`. Read it in the `osborne1-art-*` font from `tools/romfont.py` — character code **minus 65** is the glyph index, the same `sui 65` the ROM's own loader does	authored 2026-10-14
`boot-screen/o1next-art-1ng.png`	the same art rendered with the ROM's own glyphs, for reading	—
`boot-screen/o1next-art.txt`	the **O1NEXT** art: 153 × 27 cells, with *"osborne 1 next rom 2.0 2026"* and *"press return to begin"* nested in the X's counter as **text** rather than graphics (`sui 65` cannot reach them — `p` is 0x70). SHA-256 `a3617203212e193651a8dae44b3fdb030f0f5051fe629250ac2ccd0abdacd42a`	authored 2026-10-14
`boot-screen/o1next-art.png`	the same art rendered with the ROM's own glyphs	—

The measured geometry, the kerning, the byte cost of each piece, and why the long version (`O1NEXT`, 149 columns) does not fit the 128-column video buffer and is being redrawn narrower are all in `boot-screen/README.md`.

pictures/ — machine photographs

`IMG_4856.jpeg` … `IMG_4871.jpeg` — disassembly/mainboard/upgrade-board photos of the project's own machines (mainboard, double-density upgrade, ScreenPac, CoPower-88). Taken in-project for #9; these are primary sources, not copies. Further photo sets are tracked in #39.

Provenance

The two long-standing **provenance gaps are closed**: the Drive C manual and the RT-60A manual/TD0 both come from bitsavers' `bits/Osborne/Osborne1/` (`Drive_C/` and `Osborne_1_Real_Time_Clock/`), as does `OCC1_HARDDISK.IMD` and the two diagnostic disks (`_floppy_images/`). Four rows that had been recorded as in-project work or as unknown URLs were corrected on 2026-10-10.

Both remaining TODOs are closed (2026-10-10, #74):

- `copower88/swp/` — kayprojournal.com's `Swp copower 88` page; all five files matched by `File:` name through the wiki API (CoPower-88 table above).
- `nuevo/NUEVO151.BIN` — removed as a byte-identical duplicate of the tracked `tools/emulator-setup/roms/monrom-rev1.51-12.ud11` (nuevo/ table above).

No provenance TODOs remain.

The machines: a photographic survey

Status: Solved as a reference: all 16 frames in `research/pictures/` are indexed and 14 are read; two are not, and what they show is still unknown.

14 of the 16 frames were examined — `IMG_4856`–`IMG_4869`. `IMG_4870` and `IMG_4871` could not be processed by the vision pipeline, which rejects them even after an EXIF strip and a re-encode, so they are indexed but unread; they need a description or a retake.

What the 14 read frames show: the **CoPower-88 8088 board** (12 frames, `IMG_4856`–`IMG_4867`) and the **Osborne mainboard with its Z80 interposer adapter** (2 frames, `IMG_4868` and `IMG_4869`). Two findings are load-bearing elsewhere: the 8088 board is the **Kaypro variant** — silkscreen `KAYPRO-88`, software default port pair **0x7E (data) / 0x7F (status)** — and the Z80 in this machine is a plastic **NMOS Z80A, date code 8408**, which is what decides the level-shifter parts for the interposer (#11).

CoPower-88 8088 board (IMG_4856–4867)

- Silkscreen: **"KAYPRO-88"**, "(C) SWP 1983", "Lic. SWP Inc.", "COPOWER-88 REV 1.8". ⇒ The user's board is the **Kaypro variant**, consistent with software default port pair **0x7E (data) / 0x7F (status)**.
- CPU: **Fujitsu MBL8088** (NMOS 8088, ceramic DIP, 8319).
- Clock: CRYSTEK 16.000 MHz oscillator can ⟶ 8284 divides by 3 ⟶ **5.33 MHz 8088**.
- Boot ROM: 2732A labelled "SWP-8088" (the 8088-side monitor — the firmware that answers the host mailbox; a dump of this ROM would fully specify the protocol. Consider reading it when hardware is accessible — candidate for the reader hardware in #14).
- RAM daughterboard: 16 x **Mostek MK4564N-25** (64Kx1, 128 KB) soldered via SIP-style carrier above the main board.
- Glue visible matches 611-0003: 74LS138 x3 (RAS/CAS + port decode), 74LS244, 74LS373/375, 74LS645, 74LS257, 74LS175, 74LS153, 74LS32, MC74HC374 x2, 74LS161/393/174 etc.
- **Blue DIP-package IC** marked "16-2103 8140" — a **chip, not a DIP switch**, read off the physical part by the user. Almost certainly an 8-bit magnitude comparator (74LS688-class) doing the A1-A7 port-base compare from sheet 4. ⇒ The host port base is **fixed by this device / its hardwired reference, not user-switchable**. Consistent with the Kaypro software default 0x7E/0x7F. TODO: confirm the exact part number + how the reference nibble is set.
- Host ribbon: IDC at board edge (to Z80 socket adapter), plus a second 16-pin(?) white Berg-type connector (matches "BERG PINS" RDX read-strobe / expansion note on sheet 4).
- Solder side (4866): single blue bodge wire, otherwise clean.

Osborne mainboard + Z80 adapter (IMG_4868, IMG_4869)

- Mainboard marked **"24187A"** (handwritten/silk), assorted 74LS, socketed 6116 x2 (video RAM), WD1793? area partially visible.
- **The Z80 lives on the CoPower-88 interposer adapter**: a small PCB plugged into the mainboard's Z80 socket carrying
  - a plastic **Zilog Z80A, date code 8408 ⟹ NMOS** (a CMOS Z84C00 would be marked differently; also 1984 predates common CMOS Z80s)
  - 74HCT-series glue (74HCT04 visible) and pads labelled **A4 A5 A6 A7** — address lines broken out for the ribbon to the 8088 board's port decode (sheet 4 A1-A7 compare feeds the blue comparator IC; adapter passes them down)
  - ribbon cable from adapter to the KAYPRO-88 board.
- **Level-shifting conclusion for #11**: this machine is NMOS Z80A (TTL-ish outputs, high input thresholds) ⟹ use 74HCT/74AHCT toward the Z80 bus or 74LVC245 with careful thresholds; per survey (#31) the safe choice remains LVC245 host-side + HCT toward host inputs. Still confirm the other two machines before finalizing.

Open items (need user)

1. IMG_4870 / IMG_4871: what do they show? (They follow the CPU-area shots.) Please describe or retake — likely the missing piece for the adapter↔mainboard ribbon routing.
2. Which of the 3 machines is this (OCC1 label? board rev 24187A)?
3. Close-up of the blue comparator IC ("16-2103") to read the full part number and confirm the 0x7E/0x7F decode hypothesis.
4. Photos of the other two machines' Z80 areas for the NMOS/CMOS matrix in #29.

Wanted: shot list for the next open-up session

Compiled 2026-10-13, ahead of a session that opens all three machines and the upgrade boards together. Every row is something the tree is *missing*, and what it unblocks. Shoot each as its own frame, well lit and square on, file them in `research/pictures/`, and fold the findings back into this doc.

#	Shot	Why it is wanted	Serves
1	**UD11 monitor ROM label, each machine**	the 2732′s printed label usually names the revision. This is now the *only* way to identify the oldest machine (its `^D` test was never run) and it confirms the other two are 1.44.	#205, #70, #222
2	**UA15 character-generator label, each machine**	the 2716 in the chargen socket. Two images circulate and which one a machine carries tracks its BIOS revision.	#70, #24
3	**Mainboard revision silkscreen, each machine**	one board is marked `24187A`; the other two are unknown, and #39 cannot close without them.	#39, #29
4	**Z80 chip markings on the other two machines**	NMOS vs CMOS decides 74AHCT vs 74LVC in the level shifter. One machine is a plastic NMOS Z80A, date code 8408.	#39, #11, #12
5	**The Z80 socket area, interposer fitted *and* removed**	the interposer's mechanical fit, pin-1 orientation and ribbon routing. IMG_4870/4871 were meant to be this and failed the vision pipeline.	#12, #11, #39
6	**The character-generator socket and its pin-1 orientation, with the wiring around it**	gap 3 of `docs/research-gaps.md` — "char-gen socket pinout" — is **solved** from the schematic (`docs/01-mainboard-schematic.md` §4a, read UA15 off sheet 4 of 9), so this shot confirms a pin-1 orientation (#52), with its test section in a real machine	#24
10	**The [illegible] drive interface**	[illegible]	repeat #52

research/datasheets/intel-8251-usart.pdf, the 8251 datasheet

One line of description per frame is worth the minute it costs: two of the existing sixteen (`IMG_4870`, `IMG_4871`) could not be processed by the vision pipeline even after EXIF strip and re-encode, and a description is what saved them from being unreadable *and* unidentifiable.

The Portable Companion extract index

(not yet written)